Simulating air absorption in a hydraulic air compressor (HAC)

by

Stephen M. Young

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Applied Science (MASc) in
Natural Resources Engineering

The Faculty of Graduate Studies
Laurentian University
Sudbury, Ontario, Canada

THESIS DEFENCE COMMITTEE/COMITÉ DE SOUTENANCE DE THÈSE
Laurentian Université/Université Laurentienne
Faculty of Graduate Studies/Faculté des études supérieures

Title of Thesis Titre de la thèse	Simulating air absorption in a hydraulic air compressor (HAC)	
Name of Candidate Nom du candidat	Young, Stephen	
Degree Diplôme	Master of Applied Science	
Department/Program Département/Programme	MASc. Natural Resources Engineering	Date of Defence Date de la soutenance March 3, 2017

APPROVED/APPROUVÉ

Thesis Examiners/Examinateurs de thèse:

Dr. Dean Millar
(Supervisor/Directeur(trice) de thèse)

Dr. Kim Trapani
(Committee member/Membre du comité)

Dr. Ramesh Subramanian
(Committee member/Membre du comité)

Dr. Graeme Norval
(External Examiner/Examinateur externe)

Dr. Wilson Pascheto
(External Examiner/Examinateur externe)

Approved for the Faculty of Graduate Studies
Approuvé pour la Faculté des études supérieures
Dr. David Lesbarrères
Monsieur David Lesbarrères
Dean, Faculty of Graduate Studies
Doyen, Faculté des études supérieures

ACCESSIBILITY CLAUSE AND PERMISSION TO USE

Abstract

A hydraulic air compressor (HAC) is a no moving parts compressor that uses hydropower to compress air that is inducted through a Venturi effect. The solubility of gases in water increases with pressure and provides a means for the dissolved gas to bypass the air-water separator and reduce the compressed air yield of a HAC. The model developed and fully described herein extends the approach of Chen and Rice (1983) to allow for a multiplicity of gas mixtures, predicts the compressed air yield reduction from gas solubility, and predicts the compressed air yield improvement by increasing the liquid temperature and the presence of pre-dissolved salts. The model simulated multiple scenarios that adjust the parameters that are known to affect air absorption in the downcomer and returns results consistent with the expected behaviour in each scenario. One of such scenarios was the simulation of the downcomer of the Ragged Chutes HAC installation. The results were compared with the single data point of oxygen concentration in compressed air delivered by a HAC and results from an equilibrium solubility model available in the literature. Under the same conditions, the model described herein predicts a concentration of oxygen of 0.181 mol/mol, a value that is 2.3% higher than the equilibrium condition. In order to match the results of the equilibrium model, the mass diffusivities of nitrogen, oxygen, argon and carbon dioxide needed to be artificially increased from their reported values in the literature by a factor of 2000. This suggests that the solubility kinetics are important to consider in the design of a HAC when predicting the compressed air yield but if the equilibrium assumption is adopted the estimate of compressed air yield will be conservative.

Keywords: hydraulic air compressors, isothermal compression, air absorption, interphase mass transfer, solubility kinetics

Acknowledgments

I would like to thank Dean Millar for his guidance through my research, the committee members Kim Trapani, Ramesh Subramanian, Graeme Norval and Wilson Pascheto, the Ultra Deep Mine Network for funding my research, my colleagues in the HAC Demonstration Project, my colleagues at MIRARCO and the Bharti School of Engineering at Laurentian University for your assistance with my research.

Table of Contents

List of Figures

List of Tables

List of Appendices

Nomenclature

a	component of free energy of activation of diffusing solute due to cation, anion, or water, J mol^{-1}
A	area, m^2
B	bubble flux, s^{-1}
$c_{d,\infty}$	single rising bubble in infinite quiescent liquid drag coefficient, dimensionless
c_d	bubble swarm drag coefficient, dimensionless
C	concentration of species in liquid, mol m^{-3}
D	mass diffusivity of species in liquid, m^2 s^{-1}
d	diameter, m
$\overline{d}_b$	Rosin-Rammler mean bubble diameter
$d_{b,SM}$	Sauter mean bubble diameter
e	total energy per unit mass, J kg^{-1}
f	friction factor, dimensionless
F	force, N
g	gravitational acceleration, 9.8065 m s^{-2}
ΔG^*	free energy of activation of diffusing solute in electrolyte solution, J mol^{-1}
h	Planck's constant, 6.626×10^{-34} J s
H^{cp}	Henry's solubility constant defined by concentration and partial pressure, mol m^{-3} Pa^{-1}
$\check{h}_{sol}$	molar enthalpy of dissolution, J mol^{-1}
K	mass transfer coefficient, m s^{-1}
K_L	minor loss coefficient, dimensionless
k_{JD}	Jones and Dole coefficient
k_S	Sechenov constant
k	empirical constant
l	slanted height of the downcomer segment
L	length of a downcomer segment, m
LMCD	log mean concentration difference, mol m^{-3}

m	mass, kg
$\dot{m}$	mass flow rate, kg s^{-1}
M	molar mass, kg mol^{-1}
n	Number of moles of a given substance, mol
$\dot{n}$	molar flow rate, mol s^{-1}
$\hat{n}$	outward facing normal for control surface
N	empirical coefficient
N_A	Avogadro's number, 6.02×10^{23} mol^{-1}
N_{Ca}	capillary number = $U_{sg} \cdot \mu_l \cdot \sigma^{-1}$, dimensionless
$N_{Eö}$	Eötvös number defined with pipe diameter = $g \cdot d_D^2 \cdot \rho_l \cdot \sigma^{-1}$,
$N_{Eö}$'	Eötvös number defined with bubble diameter = $g \cdot d_b^2 \cdot \rho_l \cdot \sigma^{-1}$
N_{Fr}	Froude number = $U_{sg} \cdot g^{-1/2} \cdot d_D^{-1/2}$, dimensionless
N_{Fr}'	liquid Froude number = $U_{sl} \cdot g^{-1/2} \cdot d_D^{-1/2} \cdot [(\rho_l - \rho_g)/(\rho_l)]^{-1/2}$
N_{Ga}	Galilei number = $g \cdot d_D^3 \cdot \rho^2 \cdot \mu^{-2}$, dimensionless
N_{Re}	Reynolds number = $\rho \cdot U \cdot d \cdot \mu^{-1}$, dimensionless
N_{Re}'	particle Reynolds number = $(\rho_l - \rho_g) \cdot U_s \cdot d_b \cdot \mu_l^{-1}$
N_s	number of stages in a multistage compression process
P	total static pressure of fluid, Pa
p	partial pressure of a gas mixture component, Pa
$p_{sat(T),H2O}$	saturation water vapor pressure at a given temperature, Pa
q	specific heat transfer in J kg^{-1}
r	ratio involving activation energies
R	Atkinson resistance, N s^2 m^{-8} or engineering gas constant in J kg^{-1} K^{-1}
$\mathcal{R}$	universal gas constant, 8.314 J K^{-1} mol^{-1}
s	Rosin-Rammler spread parameter, dimensionless
S	surface area, m^2
t_e	exposure time of an average bubble, s
u	specific internal energy, J kg^{-1}
U	velocity, m s^{-1}
v	specific volume, m^3 kg^{-1}

$\check{v}$	molar specific volume, $m^3\ mol^{-1}$
V	volume, m^3
$\dot{V}$	volumetric flow rate, $m^3\ s^{-1}$
w	specific work, $J\ kg^{-1}$
W	weight of fluid phase in control volume, N
$\dot{W}$	power in W
x	liquid phase mole fraction, $mol\ mol^{-1}$
y	compressed air yield
z	elevation, mAD

Greek Letters

α	volume fraction of gas in a downcomer segment
β	flow direction angle measured from horizontal, rad
$\hat{\gamma}$	absolute humidity by mol, $mol\ mol^{-1}$
γ	absolute humidity by mass, $kg\ kg^{-1}$
ε	absolute roughness of the duct, m
η	efficiency
θ	wall angle measured
κ	Boltzmann's constant, $1.381x10^{-23}\ J\ K^{-1}$
μ	viscosity, Pa s
μ_n'	raw distribution moment, m^n
π	dimensionless mathematical constant, 3.14159
ρ	density, $kg\ m^{-3}$
σ	surface tension, $N\ m^{-1}$
ϕ	relative humidity of gas mixture, $Pa\ Pa^{-1}$
Φ	fraction of bubbles grater that a given diameter, dimensionless
χ	humid gas mole fraction, $mol\ mol^{-1}$
χ'	dry gas mole fraction, $mol\ mol^{-1}$
ω	mass fraction, $kg\ kg^{-1}$

Subscripts

a	dry air
avg	average
B	bulk liquid
b	bubble
c	continuous phase
cs	co-solute
D	downcomer
eq	equilibrium
g	gas phase
i	gas-liquid interface
in	at the inlet of a given domain
j	a given species being tracked (e.g. N_2, O_2)
k	a given position in the model (k = 1, 2,…)
l	liquid phase
m	mixture
out	At the outlet of a given domain
p	particle
R	riser
s	slip
sat@T	saturation at a given temperature
seg	segment
sep	separation
sg	superficial gas
sl	superficial liquid
sol	solubility
T	total
w	wall
β	component of a vector that is parallel to the flow direction

τ	shear or friction
+	cation
-	anion

Chapter 1

1. Introduction

1.1 Hydraulic air compressors (HACs)

Hydraulic air compressors (HACs) are a no-moving-parts gas compression technology developed in the 1890s by Charles H. Taylor (Schulze, 1954) which utilizes run-of-river water to induct and compress atmospheric air. They are best illustrated by the HAC installation near Cobalt, Ontario at Ragged Chutes (Figure 1.1**)**.

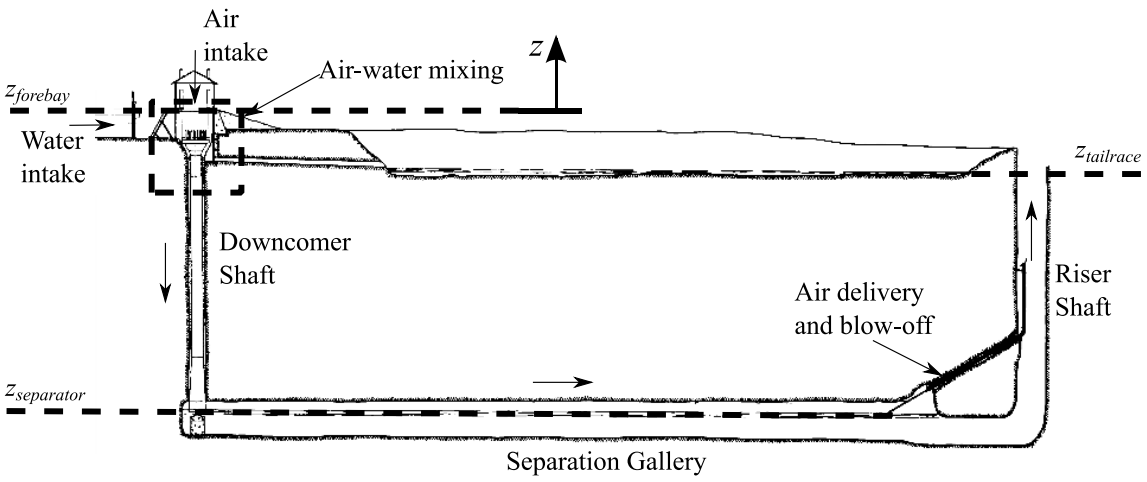


Figure 1.1 Ragged Chutes HAC installation modified from Schulze (1954)

Water collected from a river entered the system in the forebay reservoir ("Water Intake" in Figure 1.1) and was routed through the underground workings. As the water flowed down the downcomer shaft, air was inducted at the collar ("Air Intake" in Figure 1.1), creating a two-phase bubbly flow, that compressed the entrained gas as it converted the potential energy of the water

principally to pressure energy and transmitted this pressure energy to the gas. Once the two-phase bubbly flow had reached the bottom of downcomer shaft, the flow was turned horizontally to allow the air bubbles to separate from the water stream by means of the former's buoyancy ("Separation Gallery" in Figure 1.1). The compressed air was then collected over the water in a plenum and drawn off for delivery to the consumer. The water, free of bubbles, rejoined the river via the riser shaft and tailrace.

The key dimensions of these systems are the elevation difference between the water levels at the forebay ("$z_{forebay}$" in Figure 1.1) and the tailrace ("$z_{tailrace}$" in Figure 1.1) which define the energy input to the system and the elevation difference between the water level in the separator ("$z_{separator}$" in Figure 1.1) and at the tailrace defines the delivery pressure of the compressed air.

The separation gallery of the Ragged Chutes HAC was 6.1 m by 8.7 m in cross-section (20 ft. by 28.5 ft.) and 304 m (999 ft.) long. The size of the separation gallery is partly due to the air and water flow rates at the HAC but the cavern held 5,700 m^3 (200,000 ft^3) of compressed air storage as it delivered compressed air to 25 silver mines in the area (Schulze, 1954). When compressed air storage was not required, as at the Peterborough Lift Lock HAC or Clausthal HAC, HACs typically had a single shaft design with a smaller gravity air-water separator at the base as, shown in figures 1.2 and 1.3 respectively.

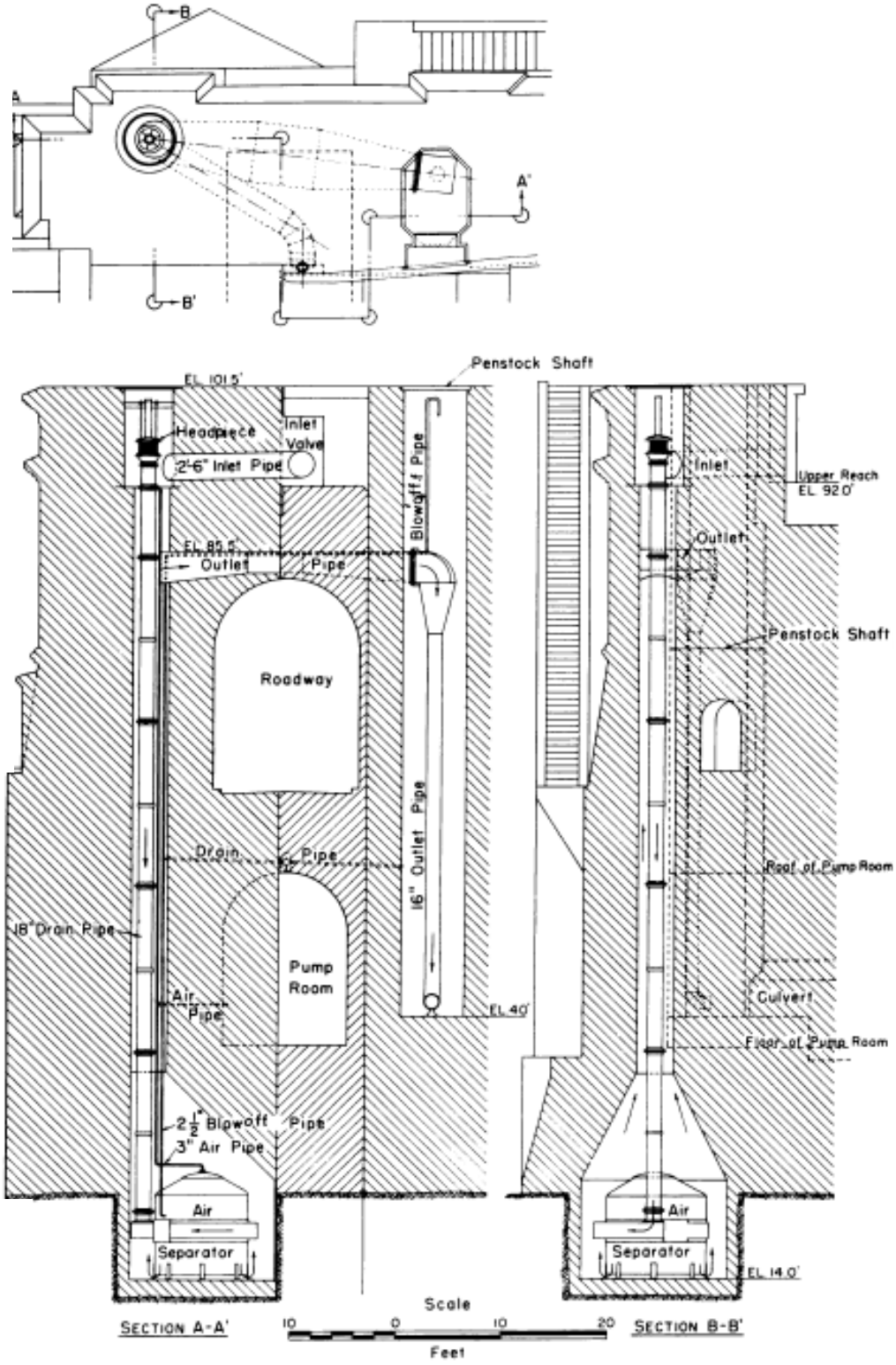


Figure 1.2 Peterborough Lift Lock HAC (Schulze, 1954)

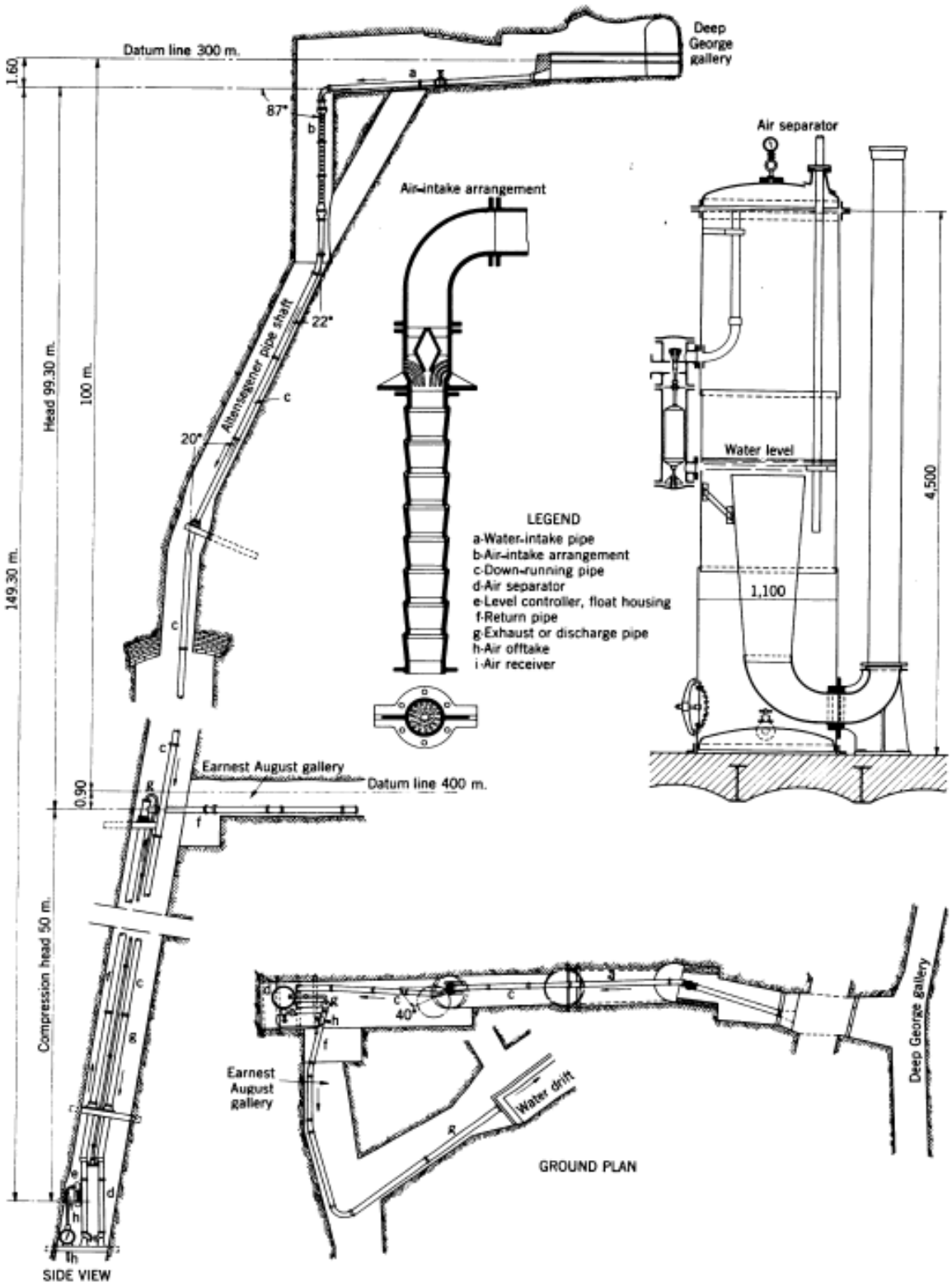


Figure 1.3 Clausthal HAC (Schulze, 1954)

Despite this technology having met its demise, HACs now show great potential to reduce the lifecycle cost of delivering compressed air due to energy efficiency gains from “near-isothermal” compression and reduced maintenance costs from their high reliability. Air being compressed by a HAC is simultaneously cooled as the heat generated from compression of the gas is transferred to the water. Since these systems typically run with a mass flow ratios of water to air of 1,000:1 to 2,000:1, and the heat capacity of water is four times that of air, and because there is a large interfacial area available for heat transfer within the two-phase bubbly downcomer flow, the resulting temperature rise of the gas is small. Thus this compression process approaches isothermal compression, the theoretically minimum work compression process. Hence it is named “near-isothermal” compression (Pavese, 2015; Pavese et al., 2016), as heat must enter the liquid phase and thus its temperature must rise a little.

HACs are reported to have a high reliability, and this is thought to be due to the few moving parts in the compression process (Schulze, 1954). The HAC installation at Ragged Chutes (Figure 1.1) ran for 72 years with only two interruptions for maintenance (Dumaresq, 2009; Moore et al., 1982).The reported and revised efficiencies of the historic HAC installations from Pavese (2015) and the known operating lives of HAC installations are summarized in tables 1.1 and 1.2 respectively.

These characteristics mean that even if the water were to be recirculated via a pump rather than using natural hydropower, there are still energy efficiency and maintenance cost savings that could be obtained by reintroducing this technology in a modern context.

Table 1.1 Revised efficiencies of selected HACs using an equilibrium solubility model modified from Pavese (2015)

No.	Year	Location	Design Parameters			Temperatures		Pressure		Mass flow rate			Yield	Efficiency			Intake/ Delivery
			Available Head [m]	Riser Depth [m]	Downcomer I.D. [m]	Water [°C]	Air [°C]	Atm [kPa]	Delivery [kPa]	Water in [kg/s]	Air in [kg/s]	Air out [kg/s]	[%]	Mech. [%]	Overall [%]	Lit. Value [%]	
1	1896	Dominion Cotton Mills, Magog, Quebec, Canada	6.58	36.55	1.13	25	27	101.3	458.5	2907.9	0.793	0.577	72.77	54.71	39.82	55	I
2	1898	Ainsworth, British Columbia, Canada	32.77	61.11	0.84	15	15	101.3	700.1	1980.2	2.949	2.643	89.64	74.1	66.42	53	I
3	1898	Dillingen Ironworks, Dillingen, Sear, Germany	1.8	12.39	1.00	15	15	101.3	222.8	867.2	0.203	0.176	86.87	86.51	75.15	79	D
4	1901	Cascade Range, Washington State, USA	13.72	60.05	0.91	15	15	101.3	689.6	1414.4	1.147	0.937	81.62	95.67	78.09		D
7	1903	Glanzenberg Mine, Nr Siegen, Germany I	40	82.58	0.10	15	15	101.3	910.4	14.2	0.026	0.023	88.79	86.01	76.37	74	D
8	1903	Glanzenberg Mine, Nr Siegen, Germany II	50	72.3	0.10	15	15	101.3	809.7	14.7	0.033	0.03	91.8	78.07	71.67	70	D
9	1903	Glanzenberg Mine, Nr Siegen, Germany III	17	72.3	0.10	15	15	101.3	809.7	14.2	0.013	0.01	79.81	91.43	72.98	70.2	D
11	1905	Holzappel, Germany	117.04	64.04	0.13	15	15	101.3	728.8	17.9	0.091	0.088	96.78	72.12	69.8	66	D
12	1906	Victoria Mine, Ontonagon County, Michigan, USA	21.34	79.55	1.52	11	12	96.5	876.3	20964	20	15.68	78.36	82.2	64.41	82	I
13	1907	Royal Mine Inspection Plant, Clausthal, Germany	99.3	50	0.22	15	15	101.3	591.2	53.3	0.21	0.203	96.73	59.18	57.24	77	D
14	1908?	Zeche Victor Rauxel Mine, Dortmund,	82	61.57	0.25	15	15	101.3	704.6	66.6	0.213	0.202	95.05	63.82	60.66	73	D
15	1909	Royal Mine Inspection Plant, Grund, Germany	36	61.85	0.30	15	15	101.3	707.3	157	0.261	0.236	90.59	75.66	68.54	88	D
16	1909	Ragged Chutes, Nr Cobalt, Ontario, Canada	16.92	83.97	2.59	21	21	101.3	923.2	29690	22.65	17.09	75.42	85.81	64.71	83	I
19	1925	Falun, Sweden	47.85	79.55	0.30	15	15	101.3	880.8	179.8	0.343	0.307	89.46	72.72	65.06	46-52	I

Table 1.2 Known operating lives of HAC installations

Location	Operating Period	Operating Life [yrs]	Notes
Dominion Cotton Mills, Magog, Quebec, Canada	1896 - 1953	57	"…the first working plant was in Magog, Quebec… successfully operated until 1953" (Dumaresq, 2009). The textile plant itself closed later in 2011 (Government of Quebec, 2013)
Ainsworth, British Columbia, Canada	1898 - 1911?	13	Only source available indicates compressor is still in use about 1911 (Schulze, 1954, p. 10)
Norwich, Conn., USA	1902 - 1929	27	"…This condition led to dismantling of the installation in 1929 to permit construction of a hydroelectric power plant" (Schulze, 1954, p. 11)
Trent Canal Lift Lock, Peterborough, Ontario, Canada	1904 - 1967	50	Still operating at the time of Schulze (1954). Officially closed in 1967 (personal communication Parks Canada, 2014)
Victoria Mine, Ontonagon County, Michigan, USA	1906 - 1921 1929 - 1930	16	Closed in 1921 due to mine closure. Reopened in 1929 to build hydroelectric dam and permanently closed in 1930 (Schulze, 1954, p. 16)
Ragged Chutes, Nr Cobalt, Ontario, Canada	1909 - 1981	72	Moore et al. (1982) states that the plant was closed permanently in September 1981.
Persberg, Sweden	1915 - 1954	39	"The Persberg compressor operated from 1915 until recently" (Schulze, 1954, p. 29)

1.2 The minimum work compression process

The thermodynamic rational for rejuvenating the HAC technology is simple. The following P-v diagram shows different processes of compressing a gas from a given pressure, P_1, to an increased pressure, P_2.

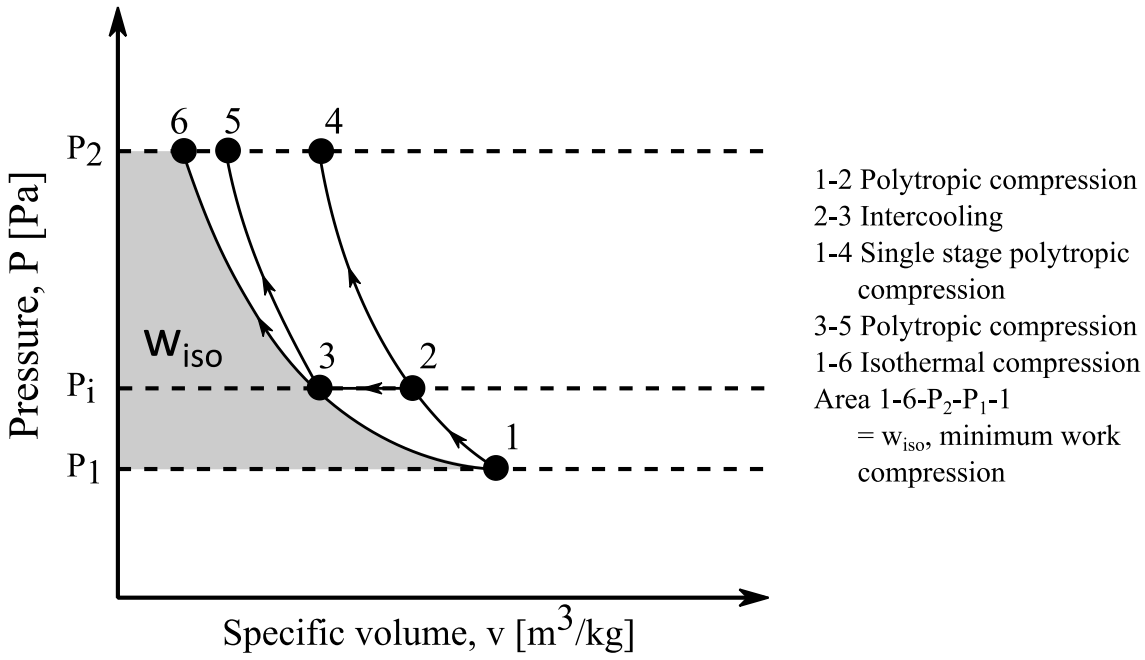


Figure 1.4 P-v diagram of gas compression

The indicated work per unit mass, w_{ind}, in the polytropic, multistage polytropic and isothermal compression can be evaluated with equations (1.1), (1.2) and (1.3) respectively (Eastop and McConkey, 1993).

$$w_{poly} = \left(\frac{n}{n-1}\right) RT_1 \left[\left(\frac{P_2}{P_1}\right)^{\frac{n-1}{n}} - 1\right] \tag{1.1}$$

$$w_{multi} = N_s \left(\frac{n}{n-1}\right) RT_1 \left[\left(\frac{P_{int}}{P_1}\right)^{\frac{n-1}{n}} - 1\right] \quad (1.2)$$

$$w_{iso} = RT \ln \frac{P_2}{P_1} \quad (1.3)$$

$$\frac{P_{int}}{P_1} = \left(\frac{P_2}{P_1}\right)^{\frac{1}{N_s}} \quad (1.4)$$

Here w_{poly} is the indicated work of polytropic compression in J/kg, n is the polytropic index, R is the engineering gas constant of the fluid being compressed in J kg^{-1} K^{-1}, T_1 is the temperature of the gas at the compressor inlet in K, w_{multi} is the indicated work of multistage polytropic compression with intercooling in J/kg, N_s is the number of stages in the multistage compression process, P_{int} is the pressure at the inlet of the intercooler in multistage compression in Pa, and w_{iso} is the indicated work of isothermal compression in J/kg. Equation (1.2) assumes that the intercooler brings the gas temperature back to the inlet temperature. This is referred to as complete intercooling.

Additionally, the heat loss from gas compression can be evaluated through the following equation (Eastop and McConkey, 1993):

$$q + w_{ind} = h_2 - h_1 = c_{p,avg}(T_2 - T_1) \quad (1.5)$$

Here w_{ind} is the indicated work input to the compressor in J/kg, h_1 and h_2 are the specific enthalpy at the compressor inlet and outlet in J/kg respectively, T_1 and T_2 are the temperatures at the compressor inlet and outlet in K respectively, $c_{p,avg}$ is the average heat capacity with constant pressure in J/kg K and q is the specific heat transfer in J/kg.

For example, if 0.075 kg/s of air is compressed mechanically from 1 bar to 9 bar the indicated work inputs and heat transfers for single stage polytropic compression, two-stage polytropic compression and isothermal compression are summarized in Table 1.3. Taking the polytropic index to be 1.3 and the engineering gas constant of air to be 286.9 J/kg K and assuming complete intercooling.

Table 1.3 Summary of indicated work input and heat losses from gas compression

Process	Indicated work input [kJ/kg]	Heat Transfer [kJ/kg]	Heat loss [kW]	Indicated power input [kW]
Single stage polytropic	236.58	-43.21	3.24	17.74
Two-stage polytropic	206.75	-37.76	2.83	15.51
Isothermal	181.65	-181.65	13.62	13.62

Most modern gas compression facilities use multistage compression. Compressing the gas in multiple stages affords intercooling. Inter and after cooling (1-2-3-5 in Figure 1.4) to reduce the amount of work required to compress the gas when compared to the basic polytropic case (1-2-4 in Figure 1.4). In a HAC the simultaneous cooling of the gas as it is compressed in the two phase bubbly flow brings the gas compression to a “near-isothermal” compression without the parasitic loads of the inter and after coolers (Pavese, 2015) and thus should provide significant energy savings even if the water is recirculated with a pump as proposed by Millar (2014).

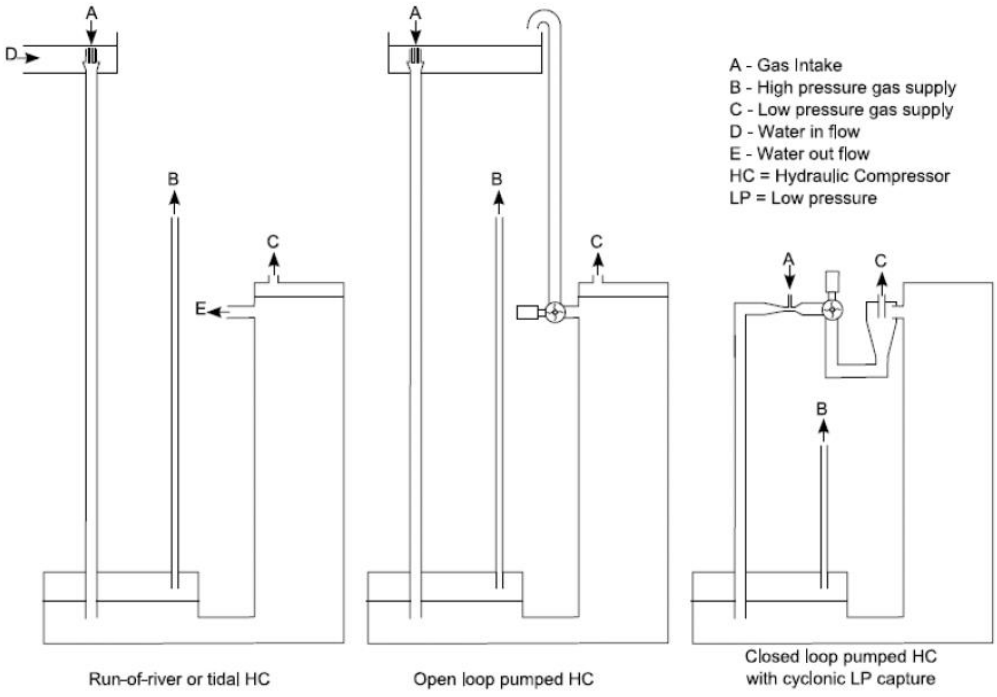


Figure 1.5 Schematic of the Taylor run-of-river HAC, open loop pumped HAC and closed loop HAC designs (Millar, 2014)

1.3 The effect of solubility on the performance of a HAC

The main limitation of the HAC arises from the behaviour of gas solubility in water (Millar, 2014). As the pressure increases, the equilibrium solubility of a given gas species also increases (Sander, 2015), which gives dissolved air a means to bypass the air-water separator. This reduces the useful compressed air yield of a HAC and results in a reduction of the overall mechanical efficiency of the system (Pavese et al., 2016). Thus when evaluating the performance of a HAC one must estimate the yield of compressed air as given with the following equations:

$$\eta_{mech} = \frac{\dot{W}_{ind}}{\dot{W}_{input}} = \frac{\dot{m}_{g,out} \cdot \int v \mathrm{d}P}{\dot{m}_{g,in} \cdot w_{12}} = y \cdot \frac{\int v \mathrm{d}P}{w_{12}} \tag{1.6}$$

$$y = \frac{\dot{m}_{g,out}}{\dot{m}_{g,in}} \tag{1.7}$$

Here η_{mech} is the overall mechanical efficiency, $\dot{W}_{ind}$ is the indicated power usefully delivered to the air in its compressed state in W, $\dot{W}_{input}$ is the power input to the HAC system in W, y is the compressed air yield, $\int v dP$ is the useful pneumatic work yielded up by the compressed air when it is consumed per unit mass in J/kg, w_{12} is the specific work input to the HAC system in J/kg, $\dot{m}_{g,out}$ is the mass flow rate of compressed gas delivered by the HAC in kg/s and $\dot{m}_{g,in}$ is the mass flow rate of gas inducted by the HAC in kg/s.

Any gas dissolved in compression and separation processes are then evolved when the static pressure is reduced in the riser flow. Schulze (1954) reported observing a milky appearance of the water discharge from the Ragged Chutes HAC installation. The evolution of gases from solution in the riser flow could provide an air lift effect which could be beneficial to the HAC system but this has yet to be proven.

Pavese (2015) and Millar (2014) estimated the yield by assuming that the gas dissolution kinetics reach equilibrium before the compressed gas is tapped off for delivery to the end user. This assumption is reasonable when considering HAC systems following the paradigm of the Ragged Chutes HAC (Figure 1.1) where the air-water contact time or the air residence time over water is on the order of hours but for HACs following the design of the Peterborough Lift Lock HAC or the Clausthal HAC (figures 1.2 and 1.3 respectively) the equilibrium may not be valid with the residence time in the separation chamber of order of 10s of seconds. Chen and Rice (1983) outlined an approach for to evaluate the compressed air yield without assuming equilibrium conditions by coupling equations for the solubility kinetics to the hydrodynamic and

psychrometric equations but only considered air to be composed of nitrogen and oxygen. Chen and Rice (1983) found that the efficiency of HACs were significantly diminished by the solubility kinetics and low-head, of the order of 0.5 m, HACs were more affected than higher-head, of the order of 3 m, HACs.

While Chen and Rice's work is seminal, it can be further improved through refinements of their hydrodynamics formulation, extension of their formulation to deal with multiplicities of gas species and extensions that account for the varying gas solubility kinetics that arise when the water contains dissolved salts.

1.4 Research objectives

As a consequence of the foregoing limitations in the state of the art, the objectives of the work described in this thesis is to develop a model which:

1. improves the coupling of the hydrodynamics with the solubility kinetics and psychrometrics of the gas compression process in the downcomer of a HAC following in the approach of Chen and Rice (1983)
2. extends the approach of Chen and Rice (1983) to include additional gas species, notably argon and carbon dioxide in the gas mixture, and to consider completely different gas mixtures, such as those that may be expected in combustion off-gases.
3. predicts the reduced compressed air yield of a hydraulic air compressor due to gas solubility, and which will verify whether or not equilibrium solubility assumption is valid.

4. predicts the compressed air yield improvement from increasing liquid temperature and/or addition of a co-solute

1.5 Thesis overview

Chapter 1 introduced and contextualized the HAC technology and outlined the objectives of the thesis.

Chapter 2 presents a review of the relevant literature sources on HACs, two-phase flow hydrodynamics, gas absorption rate evaluation and gas-liquid separators.

Chapter 3 provides an overview of the modelling methodology of HACs adopted in this work

Chapter 4 describes the formulation for the modelling of the hydrodynamics, solubility kinetics and psychrometrics of the gas compression process in the downcomer shaft.

Chapter 5 presents results from the downcomer model in multiple scenarios to test the reliability of the model which are discussed in Chapter 6.

Chapter 7 summarises the major outcomes of this work and suggests potential future work.

Chapter 2

2. Literature Review

The relevant literature on modelling the HAC technology is reviewed in this chapter and the physical principles are discussed in the context of modelling hydrodynamics, solubility kinetics and psychrometrics of the two phase bubbly flow in the downcomer of any HAC, and potentially also in the riser. But before the discourse starts, some basic definitions are presented as these feature in subsequent discussions.

Superficial velocities

In two phase flow it is common to use the superficial velocity (or volume flux) of both phases and the mixture velocity. The superficial velocity of a given phase is simply the volumetric flow rate of the phase divided by the total area available for the two-phase flow and the mixture velocity is simply the summation of the superficial velocities of both phases as shown in the following equations:

$$U_{sg} = \frac{\dot{V}_g}{A} = \frac{\dot{m}_g}{\rho_g \cdot A} \tag{2.1}$$

$$U_{sl} = \frac{\dot{V}_l}{A} = \frac{\dot{m}_l}{\rho_l \cdot A} \tag{2.2}$$

$$U_m = U_{sg} + U_{sl} \tag{2.3}$$

Here U_{sg} is the superficial gas velocity in m/s, $\dot{V}_g$ is the volumetric flow rate of the gas phase in m^3/s, U_{sl} is the superficial liquid velocity in m/s, $\dot{V}_l$ is the volumetric flow rate of the liquid phase, U_m is the mixture velocity in m/s and A is the total cross-sectional area available for the

two-phase flow in m^2. The superficial velocities are used in flow regime maps of the two phase flow and correlations for gas volume fraction and bubble diameter.

Bubble Diameter

The most common measure of a bubble's size is diameter but since bubbles in two-phase flow are not spherical an equivalent diameter is required. Clift et al. (1978) use a hydraulic equivalent diameter which is defined as the diameter of a sphere with the same density and terminal velocity of the bubble. Clift et al. (1978) also use the volume equivalent diameter which is defined with the following equation:

$$d_V = \left(\frac{6V_b}{\pi}\right)^{1/3} \tag{2.4}$$

Here d_V is the volume equivalent diameter in m and V_b is the volume of a spheroid bubble in m^3.The common diameter used in problems involving mass transfer is the Sauter diameter (Nedeltchev and Schumpe, 2011). The Sauter diameter is defined as a sphere with the same volume to surface area ratio of the spheroid bubble (Brennen, 2005). The Sauter diameter of a spheroid bubble can be determined by the following equations (Scala, 2013):

$$d_{SA} = \left(\frac{S_b}{\pi}\right)^{1/2} \tag{2.5}$$

$$d_S = \frac{d_V^3}{d_{SA}^2} \tag{2.6}$$

Here d_S is the Sauter diameter in m, d_{SA} is the surface area equivalent diameter in m and S_b is the surface area of the spheroid bubble.

2.1 HACs

A significant challenge when investigating the HAC technology is that the majority of the published performance measurements of the HAC were conducted more than 100 years ago. These performance measurements and a historical overview of these compressors compiled by Schulze (1954). The only surviving literature which reports measurements of the composition of the compressed air delivered by industrial scale HACs are from E.B.W. (1910) and McNair and Koenig (1911). Compressed air delivered by the Victoria and Ragged Chutes HAC installations was sampled at the delivery point in their respective mines and both measurements found that the air had an oxygen content of 17.7% by volume. These authors were investigating the composition of the compressed air delivered by a HAC because at this time, in the 1900s, compressed air was used for mine ventilation and simultaneous provision of pneumatic power. The mines using HAC compressed air were encountering difficulties with keeping the workings lit. Candles and, later, carbide lamps, burned less brightly and longer in mines when the ventilation was supplied by a HAC. While McNair and Koenig (1911) acknowledged that the loss of compressed air yield due to solubility would hinder the performance of the HAC, their primary concern was to investigate the difficulty with the lighting and an effect on the workers. The input energy at both Victoria and Ragged Chutes HACs was provided by freely available, renewable hydropower, so there was little cost motivation for investigations. It is now known that the principal issue is the differential solubility of the gas species in water but E.B.W. (1910) and McNair and Koenig (1911) remain as the only available literature sources of experimental data for comparison of the compressed air yield reduction predicted by HAC models.

2.1.1 Modeling the performance of a hydraulic air compressor

Most of the hydrodynamic models reported in the HAC literature neglect the effect of solubility on the performance of the HAC (Bidini et al., 1999; Rice, 1976) or when the solubility loss is accounted for, it is not computed in a fully coupled manner with the hydrodynamics (Millar, 2014; Pavese, 2015; Pavese et al., 2016). The first comprehensive hydrodynamic model of the HAC was developed by Rice (1976) which solved the flow through a HAC analytically using the conservation of mass, energy, momentum and a terminal slip velocity and numerically solved for the mass flow rate of gas inducted by the HAC. While Rice's formulation of the slip velocity equation was based upon the balance of buoyancy and drag forces, it relied upon an empirically determined initial slip velocity at the downcomer inlet. Bidini et al. (1999) simplified Rice's analysis of the downcomer and assumed it to be isothermal throughout the process and solved the flow through a HAC numerically, also utilising Rice's empirical value for the initial slip velocity at the downcomer inlet. The hydrodynamic model developed by Millar (2014) refined the approach of Bidini by not assuming isothermal conditions through the flow, solving conservation of mass, momentum, and energy equations in the water-air mixing geometry, evaluating wall friction in the downcomer by averaging the properties of both phases and solving for the initial slip velocity of the gas numerically as part of the solution procedure but does not fully couple solubility effects.

Millar (2014) did not, however, underappreciate the effect of solubility on the performance of these systems as shown by the introduction of the overall efficiency which was then expanded upon by Pavese (2015) and Pavese et al. (2016) shown in equation (1.6). Millar (2014) and Pavese (2015) revised the reported mechanical efficiencies of the eighteen previous HAC

installations to account for solubility and showed a significant reduction in the efficiency of a HAC when accounting for the reduction of compressed air yield due to solubility. The problem was that the efficiency changed significantly depending upon whether the gas flow rate was measured at the inlet or outlet of the HAC.

Gas solublity, which by definition is an equilibrium quantity, depends on pressure. Millar (2014) and Pavese (2015) both estimated the compressed air yield by assuming that the gas has reached equilibrium solubility at the HAC delivery pressure as determined by Henry's Law (see section 2.3.2), before being delivered by the HAC. This is a reasonable assumption when applied to HACs of the design like Ragged Chutes or Victoria where a large air-water separator was used for compressed air storage so that the residence time of the gas in the system is of the order of days. This may not apply to HACs following the design of the Peterborough Lift Lock installation, where there is no storage volume and the residence time in the air-water separator is on the order of seconds. In the forward case, there was sufficient time for the solubility kinetics to "complete", whereas in the latter case solubility kinetics almost certainly are still in transient.

Chen and Rice (1983) outlined a model which extended the Rice (1976) model to compute the solubility kinetics and psychrometrics in tandem with the hydrodynamics. Approximating air as a mixture of 79% nitrogen and 21% oxygen by mole, their modelling results showed that including the air absorption in the model significantly reduced the efficiency of the HAC which agrees with Millar (2014) and Pavese et al. (2016). However, Chen and Rice (1983) failed to describe in detail how the addition of the solubility kinetic and psychrometric equations impacted the previous analytical formulation. The main objective of this thesis is to develop a hydrodynamic model of downcomer flow which couples the hydrodynamics with the solubility

kinetics and psychrometrics following the approach from Chen and Rice (1983) and extend it to include argon and carbon dioxide in the gas mixture.

2.2 Two-phase flow hydrodynamics

2.2.1 Flow regime

In general, the first step to modelling two phase flow is to determine which regime the flow situated in. Of the studies that investigate the downward co-current two phase flow like that seen in the downcomer of a HAC, the main flow regimes have been established in the literature and are shown in Figure 2.1.

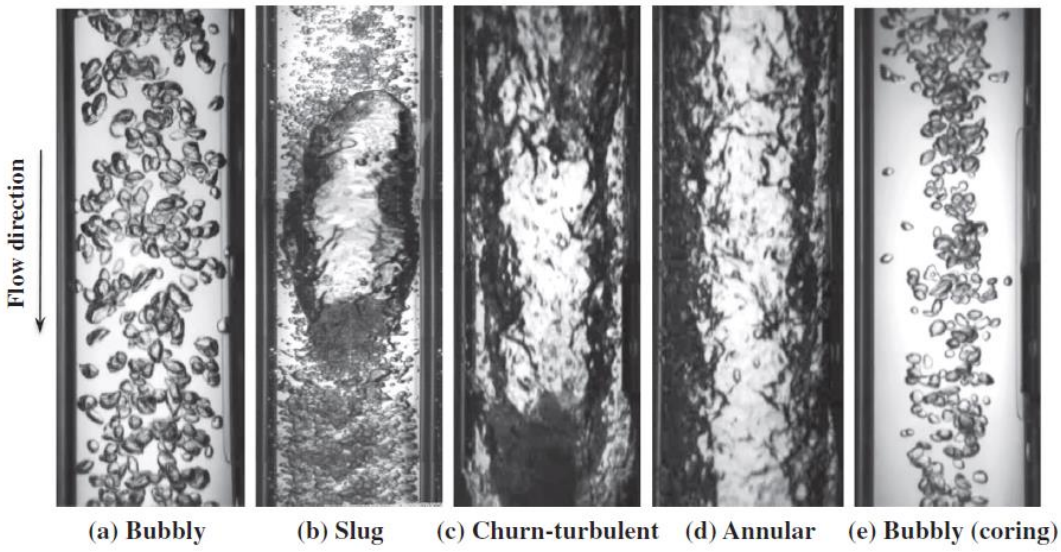


Figure 2.1 Flow regimes of vertically downward co-current two phase flow (Qiao et al., 2016).

The flow regime maps that are utilised to evaluate the expected flow regime typically use the superficial velocities of the phases or a combination of the superficial velocities and dimensionless numbers. For example, the flow regime map developed by Usui (1989), shown in

Figure 2.2, used a ratio of the superficial velocities on the horizontal axis and the liquid Froude number on the vertical axis. The liquid Froude number is determined with the following equation:

$$N_{Fr}{}' = \frac{U_{sl}}{\sqrt{g d_D \frac{(\rho_l - \rho_g)}{\rho_l}}} \tag{2.7}$$

Here U_{sl} is the superficial liquid velocity in m/s, g is the acceleration due to gravity in m/s^2, d_D is the diameter of the duct in m, and ρ_l and ρ_g are the densities of the liquid and gas phase in kg/m^3 respectively. Using the data from historical HAC installations in Table 1.1, the superficial velocities and liquid Froude numbers at the inlet and outlet of the downcomers are summarised in Table 2.1 and plotted on the flow regime map in Figure 2.2.

Table 2.1 Superficial velocity ratios and liquid Froude numbers for the historical HAC installations

No.	Year	Location	No. Downcomers	Superficial velocity ratio		Froude number	
				In	Out	In	Out
1	1896	Dominion Cotton Mills, Magog, Quebec, Canada	1	0.231	0.037	0.874	0.876
2	1898	Ainsworth, British Columbia, Canada	1	1.214	0.157	1.254	1.258
3	1898	Dillingen Ironworks, Dillingen, Sear, Germany	1	0.191	0.075	0.353	0.353
4	1901	Cascade Range, Washington State, USA	1	0.661	0.079	0.720	0.723
7	1903	Glanzenberg Mine, Nr Siegen, Germany I	1	1.493	0.147	1.829	1.837
8	1903	Glanzenberg Mine, Nr Siegen, Germany II	1	1.831	0.208	1.893	1.900
9	1903	Glanzenberg Mine, Nr Siegen, Germany III	1	0.747	0.072	1.829	1.836
11	1905	Holzappel, Germany	1	4.146	0.556	1.268	1.273
12	1906	Victoria Mine, Ontonagon County, Michigan, USA	3	0.809	0.070	0.992	0.996
13	1907	Royal Mine Inspection Plant, Clausthal, Germany	1	3.213	0.531	1.002	1.005
14	1908?	Zeche Victor Rauxel Mine, Dortmund, Germany	1	2.608	0.355	0.834	0.837
15	1909	Royal Mine Inspection Plant, Grund, Germany	1	1.356	0.175	1.246	1.251
16	1909	Ragged Chutes, Nr Cobalt, Ontario, Canada	2	0.634	0.052	0.560	0.563
19	1925	Falun, Sweden	1	1.556	0.160	1.427	1.434

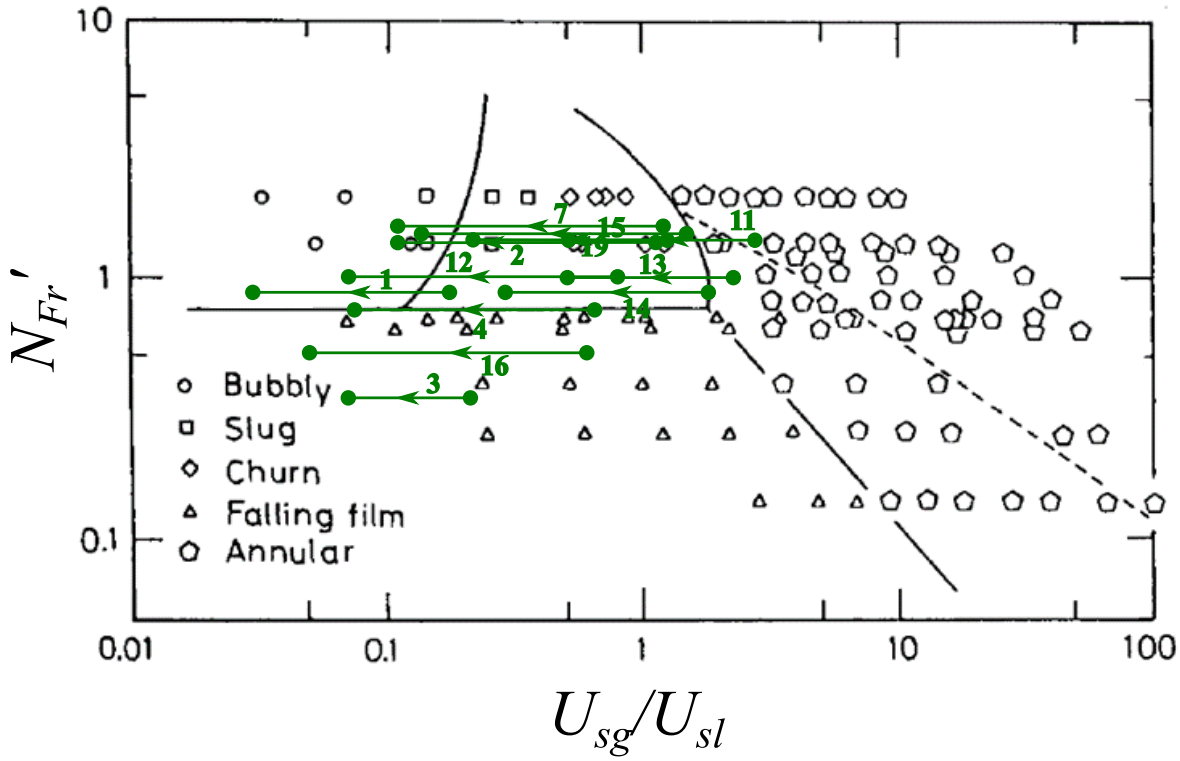


Figure 2.2 Flow regime map for downward co-current two-phase flow modified from (Usui, 1989) showing historical HAC downcomer processes (entry number of HAC as given in Table 2.1)

The two phase flow in the downcomer of a HAC typically begins in a slug flow regime which transitions to a bubbly flow regime as the superficial gas velocity is reduced from the increase in density as the flow transits the downcomer shaft.

A consensus on the boundaries of the flow regimes in the literature has not been reached (Qiao et al., 2016). Usui (1989) compared his determination of the bubbly-slug transition with previous studies (Figure 2.3) and showed a large transitional zone between the two flow regimes. Qiao et al. (2016) suggests that this may be due to the influence of the air-water mixing geometry.

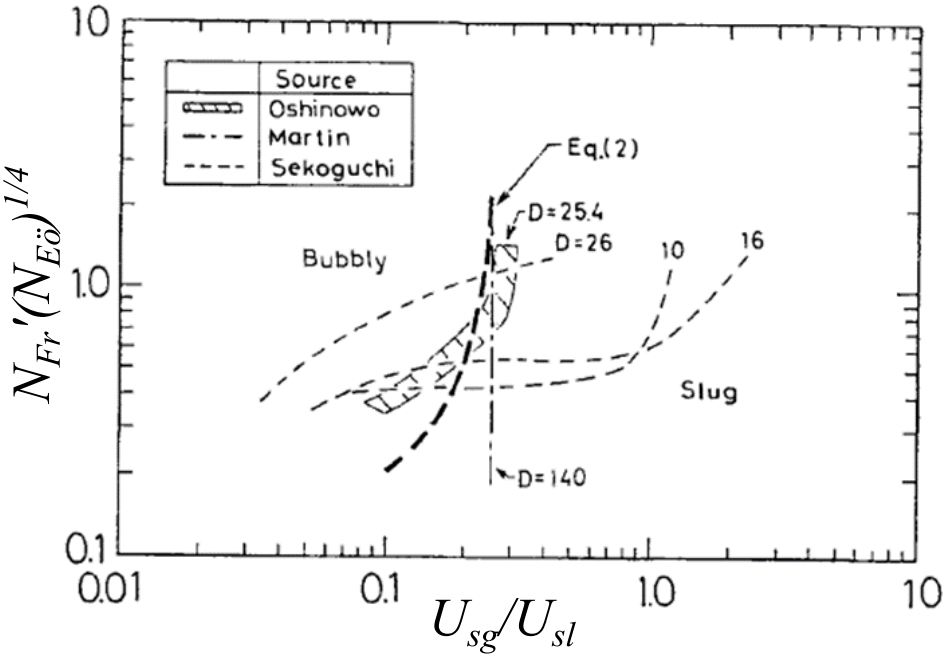


Figure 2.3 Flow regime map showing the transition from bubbly to slug flow regimes modified from Usui (1989).

In Figure 2.3, $N_{Eö}$ is the Eötvös (pronounced "Ertversh") number which is evaluated with the following equation:

$$N_{Eö} = \frac{(\rho_l - \rho_g) g d_D^2}{\sigma_l} \tag{2.8}$$

where σ_l is the surface tension of the liquid phase in N/m.

A unique phenomena of downward bubbly flow is the coring effect (Figure 2.1e) where the bubbles concentrate in the centre and rotate as they transit down the axis of the pipe. This phenomena is due to buoyant and wall repulsion forces (Qiao et al., 2016) but since the pipe diameters typically used in the flow regime studies range from 9 to 80 mm with the typical

diameter being 25.4 mm (1 inch) it is speculated that the coring phenomena is simply a scale effect and the relative magnitude of the wall repulsion forces would diminish as the diameter of the downcomer approaches the scale seen in HACs like Ragged Chutes downcomer which had a diameter of 2.59 m (102 inches) and the coring effect would not be observed.

The formulation from Millar (2014) relies upon correlations for the Sauter mean bubble diameter developed by Akita and Yoshida (1974) and Wilkinson et al. (1994) from experimental data from bubble columns. Which correlation is used depends upon whether the bubbly flow is homogenous or heterogeneous. The flow regimes found in bubble columns are shown in Figure 2.4 and the bounds of the flow regimes are shown in Figure 2.5.

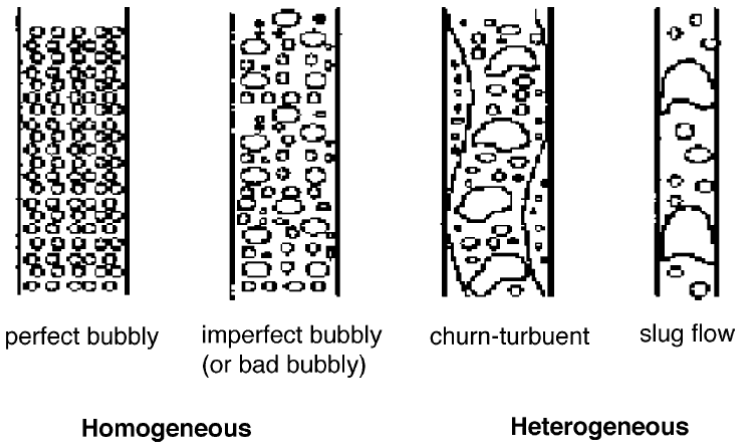


Figure 2.4 Flow regimes found in bubble columns denoting homogenous and heterogeneous bubble flows (Kantarci et al., 2005)

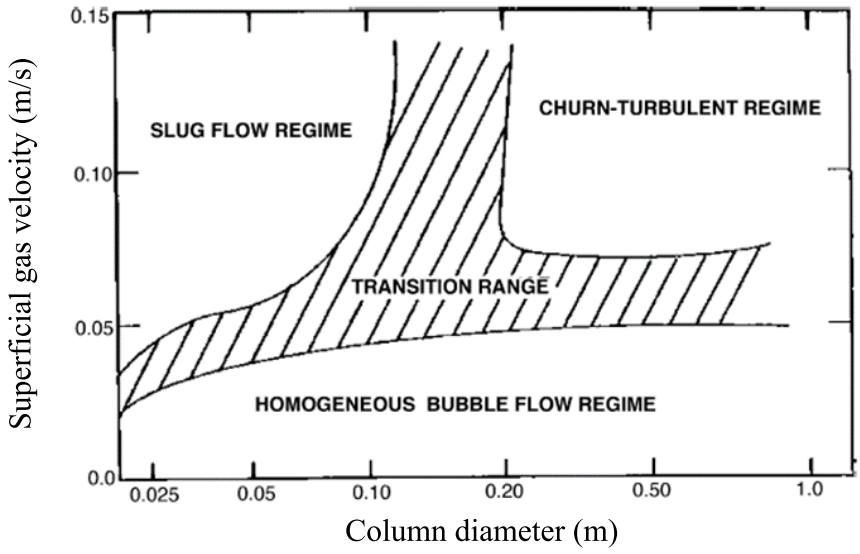


Figure 2.5 Flow regime map for bubble columns modified from (Kantarci et al., 2005)

2.2.2 Bubble size distribution

Clift et al. (1978) produced a bubble shape regime map for unassisted gravitational flow of a bubble in a liquid. It was found that using the Eötvös, particle Reynolds, and Morton numbers, the bubble shape could be determined (Figure 2.6). The Eötvös, particle Reynolds, and Morton numbers are defined using the following equations:

Eötvös:
$$N_{E\ddot{o}}{}' = \frac{g d_b^2 \rho_l}{\sigma_l} \tag{2.9}$$

Reynolds:
$$N_{Re}{}' = \frac{(\rho_l - \rho_g) \cdot U_s \cdot d_b}{\mu_l} \tag{2.10}$$

Morton:
$$N_{Mo} = \frac{g \mu_l^4}{\sigma_l^3 \rho_l} \tag{2.11}$$

Here, d_b is the diameter of a volume equivalent sphere in m, and U_s is the terminal rise velocity of the bubble. Using the data in Table 1.1 the Eötvös, Reynolds and Morton numbers were

calculated for the historical HAC installations, shown in Table 2.2, and plotted on the bubble shape regime map from Clift et al. (1978), shown in Figure 2.6.

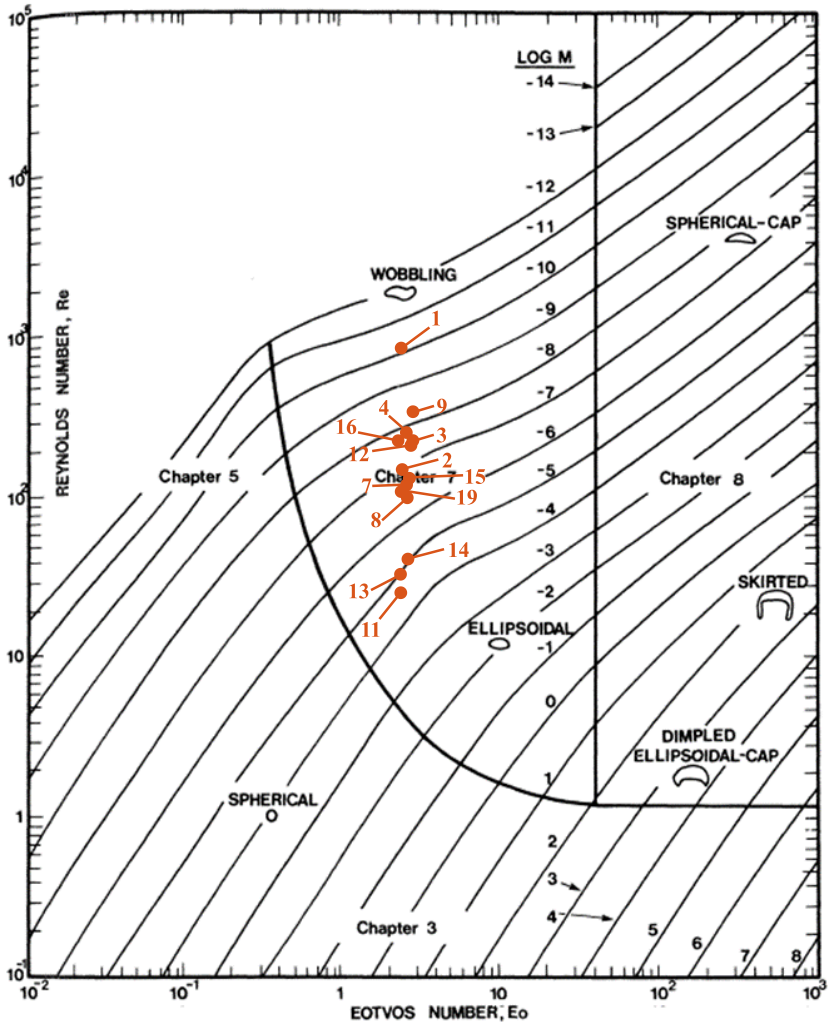


Figure 2.6 Bubble shape regime map from Clift et al. (1978), showing historical data as defined in Table 2.2

Table 2.2 Dimensionless parameters and fluid properties for bubble regime map

No.	Year	Location	Water Properties			Gas Properties				Dimensionless numbers		
			Density [kg/m3]	Viscosity [Pa-s]	Surface Tension [N/m]	Density [kg/m^3]	Superficial Velocity [m/s]	Bubble diameter [m]	Slip velocity [m/s]	$N_{Eö}'$	N_{Re}'	log(N_{Mo})
1	1896	Dominion Cotton Mills, Magog, Quebec, Canada	997.05	8.90E-04	0.07	1.18	0.67	4.20E-03	0.19	2.4	887	-10.78
2	1898	Ainsworth, British Columbia, Canada	999.10	1.14E-03	0.07	1.23	4.36	4.27E-03	0.04	2.4	153	-10.38
3	1898	Dillingen Ironworks, Dillingen, Sear, Germany	999.10	1.14E-03	0.07	1.23	0.21	4.54E-03	0.06	2.7	231	-10.38
4	1901	Cascade Range, Washington State, USA	999.10	1.14E-03	0.07	1.23	1.43	4.37E-03	0.07	2.5	249	-10.38
7	1903	Glanzenberg Mine, Nr Siegen, Germany I	999.10	1.14E-03	0.07	1.23	2.70	4.32E-03	0.04	2.5	135	-10.38
8	1903	Glanzenberg Mine, Nr Siegen, Germany II	999.10	1.14E-03	0.07	1.23	3.43	4.30E-03	0.03	2.5	101	-10.38
9	1903	Glanzenberg Mine, Nr Siegen, Germany III	999.10	1.14E-03	0.07	1.23	1.35	4.38E-03	0.08	2.6	324	-10.38
11	1905	Holzappel, Germany	999.10	1.14E-03	0.07	1.23	5.86	4.25E-03	0.01	2.4	25	-10.38
12	1906	Victoria Mine, Ontonagon County, Michigan, USA	999.61	1.27E-03	0.07	1.18	3.10	4.44E-03	0.06	2.6	217	-10.21
13	1907	Royal Mine Inspection Plant, Clausthal, Germany	999.10	1.14E-03	0.07	1.23	4.68	4.27E-03	0.01	2.4	33	-10.38
14	1908?	Zeche Victor Rauxel Mine, Dortmund, Germany	999.10	1.14E-03	0.07	1.23	3.43	4.29E-03	0.01	2.5	42	-10.38
15	1909	Royal Mine Inspection Plant, Grund, Germany	999.10	1.14E-03	0.07	1.23	2.92	4.31E-03	0.04	2.5	136	-10.38
16	1909	Ragged Chutes, Nr Cobalt, Ontario, Canada	998.00	9.78E-04	0.07	1.20	1.79	4.20E-03	0.05	2.4	220	-10.63
19	1925	Falun, Sweden	999.10	1.14E-03	0.07	1.23	3.84	4.28E-03	0.03	2.4	117	-10.38

Due to the high turbulence in downward co-current bubble flow, the bubbles tend to be spheroid or ellipsoidal (see figures 2.7 and 2.8). Bhagwat and Ghajar (2012) found that with a given gas superficial velocity and increasing liquid superficial velocity the bubbles tend to become more numerous, reduce in size and more spherical and, with a given liquid superficial velocity and increasing gas superficial velocity, the bubbles become more ellipsoidal and larger in size.

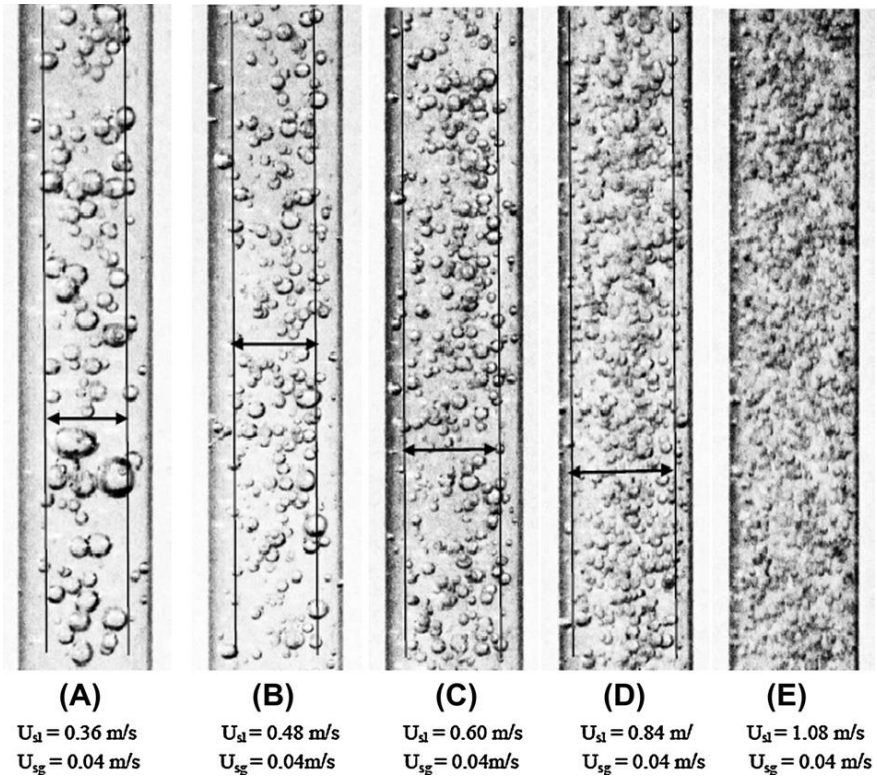


Figure 2.7 Downward co-current bubbly flow with increasing liquid superficial velocity highlighting the coring effect (Bhagwat and Ghajar, 2012)

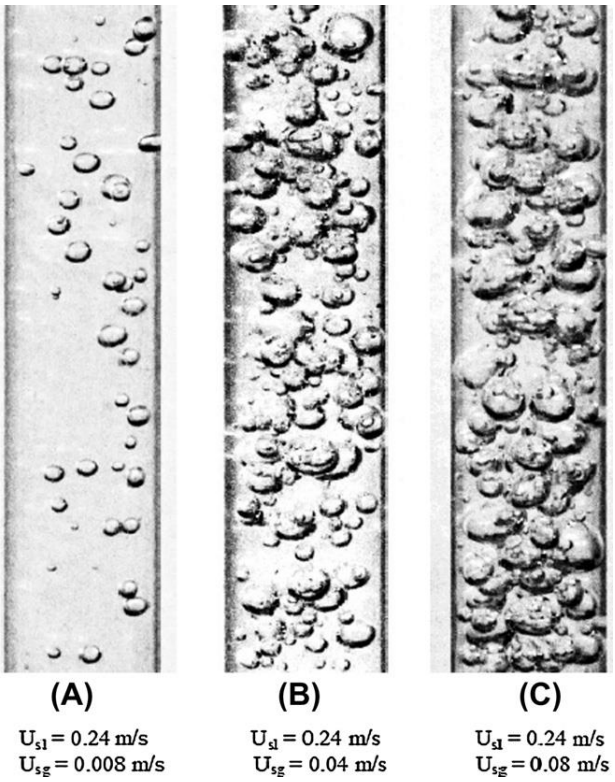


Figure 2.8 Downward co-current bubbly flow with increasing gas superficial velocity (Bhagwat and Ghajar, 2012)

The bubbles in two phase bubbly flow do not have a uniform shape or size as shown in Figure 2.7 but their size, follow a log-normal distribution (Akita and Yoshida, 1974). The size distribution is commonly represented with the Rosin-Rammler distribution shown with the following equation (Johansen et al., 2000; Laleh, 2010):

$$\Phi = 1 - \exp\left[-\left(\frac{d_b}{\bar{d}}\right)^s\right] \tag{2.12}$$

Here d_b is the diameter of a given bubble in m, Φ is the fraction of bubbles with a diameter less than d_b, $\bar{d}$ is Rosin-Rammler mean diameter in m and s is a spread parameter. The Rosin-Rammler mean diameter, $\bar{d}$, is a statistical mean which corresponds to the bubble diameter with 63% of the bubbles having a diameter less than $\bar{d}$. The Rosin-Rammler spread parameter, s, must

be determined empirically. From bubble count data or using the Rosin-Rammler distribution, the Sauter mean diameter of the bubble population is evaluated using the following equations (Akita and Yoshida, 1974):

$$\mu_n' = \int_0^\infty x^n f(x)\,\mathrm{d}x = \int_0^1 x^n\,\mathrm{d}\Phi \approx \sum x_j^n \cdot \Delta\Phi_j \tag{2.13}$$

$$d_{b,SM} = \frac{\mu_3{}'}{\mu_2{}'} \tag{2.14}$$

Here $\mu_n{}'$ is the nth order moment of the distribution. As an example, this approach is used to calculate the Sauter mean bubble diameter using data from Akita and Yoshida (1974) and shown in Table 2.3.

Table 2.3 Illustrative example of calculating Sauter mean bubble diameter from a bubble size distribution (Akita and Yoshida, 1974)

Range [mm]		x_j [mm]	No. bubbles	$\Delta\Phi_j$ [%]	Φ [%]	$\Delta\mu 2'$ [mm]	$\Delta\mu 3'$ [mm]
0.9	1	0.949	260	8.225	8.23	0.0740	0.07
1.1	1.5	1.28	450	14.24	22.47	0.2350	0.30
1.6	2	1.79	416	13.16	35.63	0.4211	0.75
2.1	2.5	2.29	452	14.3	49.93	0.7508	1.72
2.6	3	2.79	454	14.36	64.29	1.1201	3.13
3.1	4	3.52	479	15.15	79.44	1.8786	6.62
4.1	5	4.53	241	7.624	87.06	1.5629	7.08
5.1	6	5.53	141	4.461	91.52	1.3651	7.55
6.1	8	6.99	145	4.589	96.11	2.2394	15.64
8.1	10	9.00	58	1.835	97.94	1.4864	13.38
10.1	15	12.3	40	1.265	99.21	1.9165	23.59
15.1	20	17.4	18	0.5694	99.78	1.7196	29.88
20.1	25	22.4	3	0.09491	99.87	0.4769	10.69
27	0	27.0	1	0.03164	99.9	0.2307	6.23
27.4	0	27.4	1	0.03164	99.94	0.2375	6.51
30	0	30.0	1	0.03164	99.97	0.2848	8.54
30.3	0	30.3	1	0.03164	100	0.2905	8.80
			Σ=3161	**Σ=100.00**		**Σ=16.29**	**Σ=150.48**

Applying equation (2.14) gives a Sauter mean diameter of 9.24 mm.

If bubble distribution data is unavailable, the Sauter mean diameter can be estimated using empirical correlations. The correlation from Wilkinson et al. (1994) is most commonly used (Nedeltchev and Schumpe, 2011):

$$N_{E\ddot{o}}{}' = 8.8 \cdot N_{Ca}^{-0.04} \cdot N_{Mo}^{0.12} \cdot \left(\frac{\rho_l}{\rho_g}\right)^{0.22} \tag{2.15}$$

$$N'_{E\ddot{o}} = \frac{g\rho_l d_{b,SM}^2}{\sigma_l} \tag{2.16}$$

$$N_{Ca} = \frac{U_{sg}\mu_l}{\sigma_l} \tag{2.17}$$

$$N_{Mo} = \frac{g\mu_l^4}{\sigma_l^3 \rho_l} \tag{2.18}$$

Here $N_{E\ddot{o}}{}'$ is the Eötvös number defined with bubble diameter, N_{Ca} is the capillary number, and N_{Mo} is the Morton number. This correlation is used by Millar (2014) for heterogeneous bubble flow and the correlation from Akita and Yoshida (1974) is used for homogeneous bubble flow:

$$\frac{d_{b,SM}}{d_D} = 26 \cdot N_{E\ddot{o}}^{-0.5} \cdot N_{Ga}^{-0.12} \cdot N_{Fr}^{-0.12} \tag{2.19}$$

$$N_{E\ddot{o}} = \frac{g d_D^2 \rho_l}{\sigma_l} \tag{2.20}$$

$$N_{Ga} = \frac{g d_D^3 \rho_l^2}{\mu_l^2} \tag{2.21}$$

$$N_{Fr} = \frac{U_{sg}}{\sqrt{g d_D}} \tag{2.22}$$

Here $N_{E\ddot{o}}$ is the Eötvös number defined by duct diameter, N_{Ga} is the Galilei number and N_{Fr} is the Froude number defined by superficial gas velocity and duct diameter.

2.2.3 Drag

In the HAC downcomer, the gas phase is able to resists the buoyant force due to the drag imposed on it by the liquid phase and hence is a key phenomenon required for the HAC to function. The modelling of the drag force on a bubble relies on the evaluation of a drag coefficient of a free settling/rising particle in an infinite medium such that the drag force can be evaluated by the following equation:

$$F_d = \frac{\pi}{8} \cdot c_{d,\infty} \cdot d_b^2 \cdot \rho_l \cdot U_s^2 \tag{2.23}$$

Here $c_{d,\infty}$ is the drag coefficient of a bubble in an infinite medium, d_b is the bubble diameter in m and U_s is the terminal slip velocity in m/s. There are numerous studies on the drag forces of free settling particles (e.g. Brown and Lawler, 2003; Cheng, 2009) and there are several empirical relationships for the drag coefficient (Brown and Lawler, 2003; Cheng, 2009; Karamanev and Nikolov, 1992). Karamanev and Nikolov (1992) observed that free rising particles of a density less than 300 kg/m^3 did not follow the standard drag curve but the curve flattened out at 0.95 as shown in the following figure:

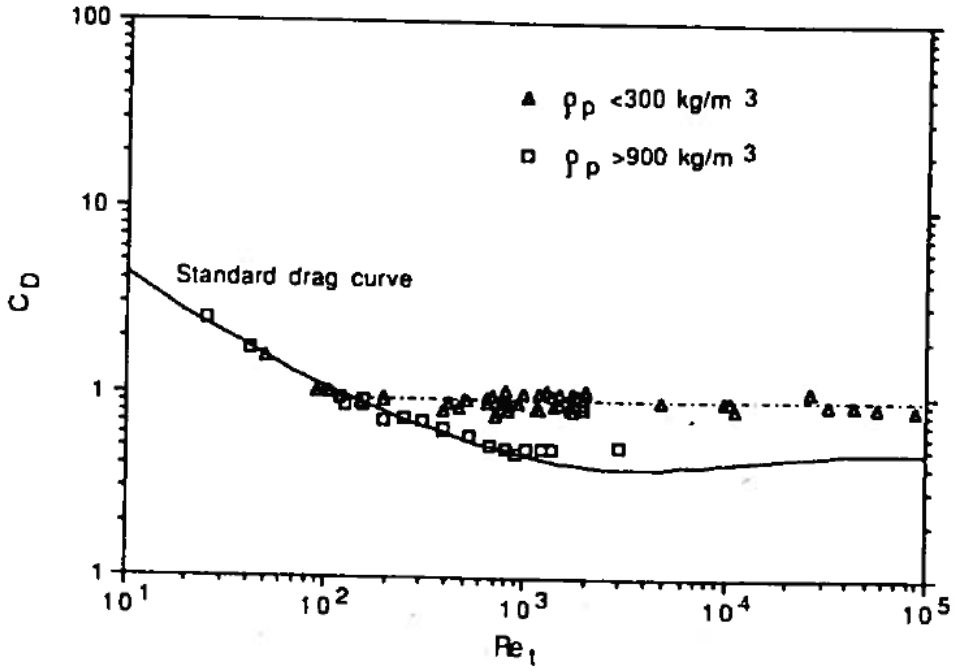


Figure 2.9 The drag coefficient of a free rising particle in an infinite medium (C_D or $c_{d,\infty}$) versus particle Reynolds number (Re_t or N_{Re}')

It is uncertain if this applies to bubbles in the downcomer or riser flow because the bubbles are not free rising in either flow. Since the liquid inertia would overwhelm any gravitational effects, it is speculated that the bubbles in the downcomer and riser flows would follow the standard drag curve and the drag coefficient would be dictated by the slip velocity of the gas.

For the purposes of comparison, the drag coefficient is evaluated adopting the approach from Chen and Rice (1983) using the relationship from Rowe and Henwood (1961):

$$c_{d,\infty} = \begin{cases} \frac{24}{N_{Re}'}\left(1 + 0.15{N_{Re}'}^{0.687}\right) & if\ N_{Re}' < 1000 \\ 0.44 & if\ N_{Re}' \geq 1000 \end{cases} \tag{2.24}$$

$$N_{Re}' = \frac{(\rho_l - \rho_g) \cdot U_s \cdot d_b}{\mu_l} \tag{2.25}$$

Because the bubbles in a HAC downcomer flow are not sparse, the drag coefficient must be modified to account for the influence of the surrounding bubbles. Chen and Rice (1983) adopted the following equation:

$$c_d = c_{d,\infty}(1-\alpha)^{-2N} \tag{2.26}$$

Here α is the gas volume fraction of the two phase flow, N a coefficient found to be 2.35 by Rowe and Henwood (1961), and c_d is the drag coefficient of the bubble swarm.

2.2.4 Volume fraction

Evaluating the drag coefficient in the downcomer flow necessitates the evaluation of the gas volume fraction, however, Chen and Rice (1983) did not articulate how this was evaluated in their model. Bhagwat and Ghajar (2012) performed a review of the correlations found in the literature for vertically downward two phase flow and summarized which correlations were the most accurate in the volume fraction ranges of $0 < \alpha \leq 0.25$, $0.25 < \alpha \leq 0.50$, $0.5 < \alpha \leq 0.75$ and $0.75 < \alpha \leq 1.00$. Based upon the expected flow regimes (Figure 2.3), it is expected that the HAC downcomer flow will have a volume fraction in the range of $0 < \alpha \leq 0.5$ and it was found that the correlation from Cai et al. (1997) had an RMS error of 11.5% in the $0 < \alpha \leq 0.25$ range and 9.5% in the $0.25 < \alpha \leq 0.5$ range and hence is presented in the following equation for adoption within the formulation herein:

$$\alpha = \frac{U_{sg}}{1.15U_m \pm 0.345\sqrt{g d_D\left(1-\frac{\rho_g}{\rho_l}\right)}} \tag{2.27}$$

However, all the correlations reviewed by Bhagwat and Ghajar (2012) performed poorly, i.e., an RMS error of greater than 50%, when applied to vertically upward two phase flow in the volume

fraction range of $0 < \alpha \leq 0.25$. The implications of this is that when modelling the riser flow with imperfect separation efficiency in the air-water separator there could be significant error in the determination of the volume fraction. Since the volume fraction is used in the evaluation of the total interfacial area, this could add significant error to the solubility kinetic equations depending upon the magnitude of the volume fraction of gas in the riser flow.

2.3 Gas absorption rate

The foundational study on gas absorption rate was conducted by Whitman (1923) who gave the first description of the two-film (or two-resistance) theory. Two-film theory states that the molar transfer rate across a gas liquid interface is a function of the rate that it travels to the interface from the bulk gas phase and from the interface into the bulk liquid (Figure 2.10).

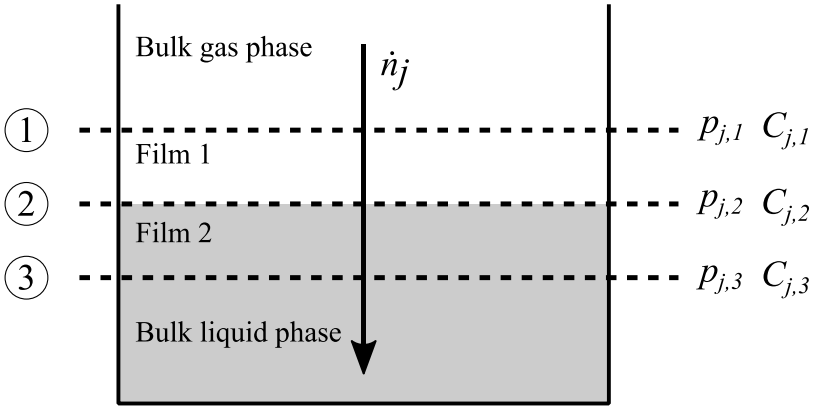


Figure 2.10 Two-film theory modified from Whitman (1923). Here $p_{j,k}$ and $C_{j,k}$ are the partial pressures and molar concentrations of species j at position k respectively

Whitman (1923) expressed two-film theory with the following equation:

$$\dot{n}_j = K_g\left(p_{j,1} - p_{j,2}\right)A_i = K_l\left(C_{j,2} - C_{j,3}\right)A_i \tag{2.28}$$

Here *1* is the bulk gas phase, *2* is the gas-liquid interface, *3* is the bulk liquid phase, $p_{j,k}$ is the partial pressure of species *j* at a given location *k* in Pa, $C_{j,k}$ is the concentration of species *j* at a given location *k* in mol/m^3, K_g is the gas phase mass transfer coefficient, K_l is the liquid phase mass transfer coefficient and A_i is the interfacial area in m^2. The term of $(p_{j,1} - p_{j,2})$ is called the "driving force" for gas absorption in the gas phase and the term of $(C_{j,2} - C_{j,3})$ is the "driving force" for gas absorption in the liquid phase.

If the limiting resistance is in the gas phase, the overall mass transfer rate can be described using the gas phase parameters of equation (2.28) only. Similarly, if the limiting resistance is in the liquid phase, the overall mass transfer can be described using the liquid phase parameters of equation (2.28) only. To illustrate the concept of limiting resistance, Whitman (1923) provided the example of the absorption of hydrochloric acid (HCl). Since the vapour pressure, i.e. $p_{HCl,2}$, of HCl in the gas phase is low, but not negligible, for liquid phase concentrations of approximately 250 g/L (42 Pa at 25°C), the driving force in the gas phase is lower than in the liquid phase. Consequently, the liquid phase will absorb gas as rapidly as it can diffuse from the bulk gas phase to the gas-liquid interface. In the case of the HAC downcomer, since the gas phase is comprised of small bubbles, the rate at which the gas solutes diffuse through the bubbles can be neglected and gas absorption rate would be limited by the rate that gas diffuses from the gas-liquid interface to the bulk fluid.

Two-film or two-resistance theory assumes that there is no resistance to molecular diffusion at the gas-liquid interface. Consequently, gas solute concentration in the liquid at the gas liquid interface is always at equilibrium with the partial pressure of the gas in the gas phase. Henry's

law is used to convert the partial pressure of the species in the gas phase to its concentration in the liquid phase at the interface (Whitman, 1923).

The absorption rate can also be expressed in terms of the overall potential provided the concentrations and partial pressures in the bulk gas and liquid phases (e.g. through Henry's law).

$$\dot{n}_j = K_P\left(p_{j,1} - p_{j,3}\right)A_i \tag{2.29}$$

$$\dot{n}_j = K_C\left(C_{j,1} - C_{j,3}\right)A_i \tag{2.30}$$

Here K_p is the overall mass transfer coefficient in pressure terms, and K_C is the overall mass transfer coefficient in concentration terms. Whitman (1923) cautioned that many cases cannot be handled by the simplification of direct proportionality between partial pressure and concentration (e.g. if the deviation from Henry's law is appreciable).

Higbie (1935) investigated the liquid film resistance in simulated industrial conditions and formulated a relationship for the liquid phase mass transfer coefficient.

$$K_l = 2\sqrt{\frac{D}{\pi t_e}} \tag{2.31}$$

Here D is the diffusion constant for a given gas species in the liquid phase in m^2/s and t_e is the time of exposure of the gas and liquid in s. The time of exposure is the period required to reset the concentration gradient in the liquid and is given by the following equation:

$$t_e = \frac{\text{surface area of bubble}}{\text{rate of surface formation}} = \frac{\pi d_B^2}{\pi \mathrm{d_B} U_s} = \frac{d_b}{U_s} \tag{2.32}$$

Nedeltchev & Schumpe (2011) recommend that the Higbie relation is appropriate for a first approximation of the mass transfer coefficient which should be refined later through developing

correlations from experimental data. Since this is a first approximation of the air absorption in a HAC, the Higbie relationship is adopted in this formulation.

2.3.1 Diffusivity of gases in liquids

Evaluating the absorption rate of a given gas species requires the evaluation of the species' mass diffusivity constant in water (see equation (2.31)). The mass diffusivity is a proportional constant between the mass transfer rate and the spatial concentration gradient derived from Fick's law (Higbie, 1935):

$$\frac{\partial C}{\partial t} = D \frac{\partial^2 C}{\partial x^2} \tag{2.33}$$

Chen and Rice (1983) did not report the values of mass diffusivity adopted in their model. Perry and Green (1999) and Himmelblau (1964) tabulate mass diffusivity constants of gases in pure water at atmospheric pressure and 25°C. In context of the HAC, only the mass diffusivities of the major components of atmospheric air are required (i.e. nitrogen, oxygen, argon, and carbon dioxide).

Table 2.4 Mass diffusivity values of gas species in pure water at 25°C and 101,325 Pa (Young et al., 2016)

Species	$D_{j,l,0}$ [$m^2\ s^{-1}$]	Source
Nitrogen (N_2)	1.90×10^{-9}	(Perry and Green, 1999)
Oxygen (O_2)	2.50×10^{-9}	(Perry and Green, 1999)
Argon (Ar)	1.46×10^{-9}	(Himmelblau, 1964)
Carbon dioxide (CO_2)	1.96×10^{-9}	(Perry and Green, 1999)

Perry and Green (1999) recommend using the Stokes-Einstein equation to evaluate the mass diffusivity at other temperatures and pressures.

$$\frac{D_{j,l}T}{\mu_l} = \text{constant} \tag{2.34}$$

Here $D_{j,l}$ is the diffusivity of species j in water in m^2/s, T is the temperature in K and μ_l is the viscosity of water in Pa s. The mass diffusivity is dependent more upon temperature because the viscosity of water does not change significantly with pressure.

When evaluating the air absorption rate in to an aqueous electrolyte solution (AES) rather than pure water, the mass diffusivities need to be adjusted for the increased resistance provided by the presence of the co-solute (Ratcliff and Holdcroft, 1963). The methods used to estimate the mass diffusivity of a gas solute in an AES in the literature are semi-empirical correlations and can be placed into three categories: i) viscosity methods, ii) salt concentration methods and iii) Erying theory methods.

Viscosity method

An approach to estimate the diffusivity of a gas solute in an AES proposed by Ratcliff and Holdcroft (1963) is to correlate the ratio of the diffusivity of the gas in pure water and in the AES to the ratios of the viscosities:

$$\frac{D_{j,l}}{D_{j,H2O}} = \left(\frac{\mu_l}{\mu_{H2O}}\right)^{-N} \tag{2.35}$$

Here $D_{j,l}$ is the mass diffusivity of gas solute j in the AES, $D_{j,H2O}$ is the mass diffusivity of gas solute j in pure water, μ_l is the viscosity of the AES, μ_{H2O} is the viscosity of pure water and N is an empirical coefficient. From diffusivity and viscosity ratio experiments (shown in , Ratcliff and Holdcroft (1963) recommended a value of 0.637.

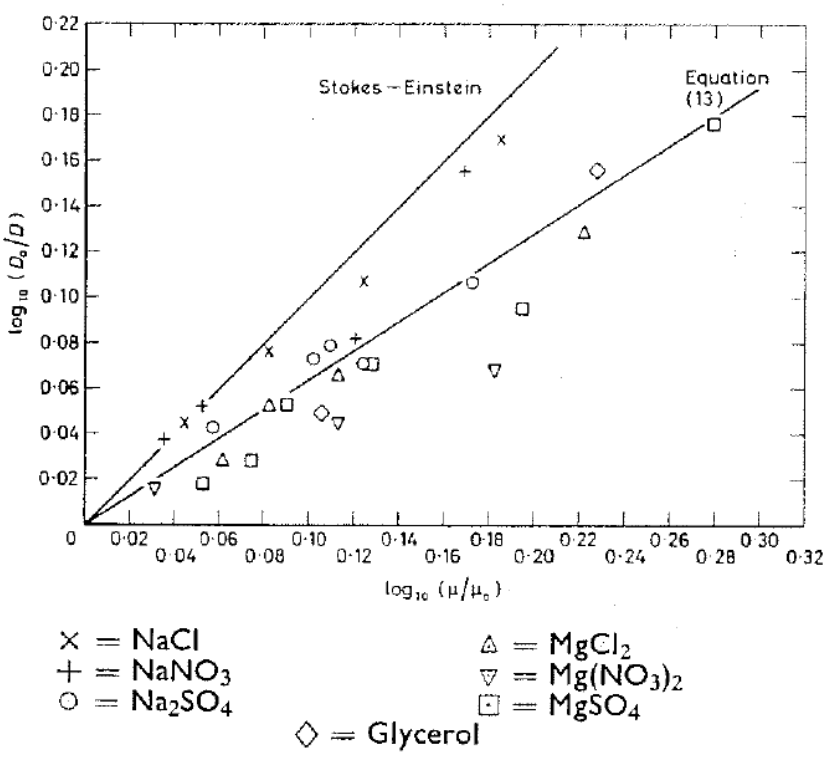


Figure 2.11 Logarithmic plot of diffusivity and viscosity ratios of pure water and AES (Ratcliff and Holdcroft, 1963)

Salt concentration method

Several sources in the literature correlate the mass diffusivity with electrolyte concentration (Himmelblau, 1964; Jamnongwong et al., 2010; Ratcliff and Holdcroft, 1963)

$$\frac{D_{j,l}}{D_{j,H2O}} = 1 - k \cdot \sqrt{C_{cs}} \tag{2.36}$$

$$\frac{D_{j,l}}{D_{j,H2O}} = 1 - k \cdot C_{cs} \tag{2.37}$$

$$\frac{D_{j,l}}{D_{j,H2O}} = \frac{k_{JD}}{1+r} \cdot C_{cs} + 1 \tag{2.38}$$

$$\frac{D_{j,l}}{D_{j,H2O}} = 1 - k \cdot C_{cs}^{N} \tag{2.39}$$

$$\log \frac{D_{j,l}}{D_{j,H2O}} = -k \cdot C_{cs} \tag{2.40}$$

Here k and N are empirical coefficients, k_{JD} is the Jones-Dole viscosity coefficient (Jones and Dole, 1929) and r is a ratio of the perturbation of activation energies for transport of the solvent and ions (Ratcliff and Holdcroft, 1963).

Eyring theory method

Akita (1981) outlined an approach to estimate the mass diffusivity of gases in electrolyte solutions based upon Eyring theory which is set out in the following equations:

$$\check{v} = \frac{1}{\sum C_{j+} + \sum C_{j-} + C_{H2O}} \tag{2.41}$$

$$C_{H2O} = \frac{\rho_l - \sum M_{j+} C_{j+} - \sum M_{j-} C_{j-}}{M_{H2O}} \tag{2.42}$$

$$\Delta G^{*} = \sum \Delta a_{+} x_{j+} + \sum \Delta a_{j-} x_{j-} + a_{H2O} \tag{2.43}$$

$$a_{H2O} = -\mathcal{R}T \ln \left[\frac{D_{j,H2O}}{T} \cdot \frac{h}{\kappa} \cdot \left(\frac{\check{v}}{N_A} \right)^{-2/3} \right] \tag{2.44}$$

$$D_{j,l} = \left(\frac{\kappa}{h} \right) \left(\frac{\check{v}}{N_A} \right)^{2/3} T \exp \left[-\frac{\Delta G^{*}}{\mathcal{R}T} \right] \tag{2.45}$$

Here h is Planck's constant, κ is the Boltzmann's constant, N_A is Avogadro's number, $\check{v}$ is the molar volume of the solution in mol/m^3, T is the liquid temperature in K, ΔG^{*} is the free energy

of activation of diffusing solute in electrolyte solution in J/mol, Δa_{j+} is the component of free energy of activation of diffusing solute due to the j^{th} cation less that of pure water and Δa_{j-} is the component of free energy of activation of diffusing solute due to the j^{th} anion less that of pure water. Akita (1981) also evaluated and tabulated the values of Δa_{j+} and Δa_{j-} for various cations and anions from his experimental work and a selection of which are shown in Table 2.5.

Table 2.5 Values of Δa_{j+} and Δa_{j-} for selected cations and anions from Akita (1981)

Cation	Δa_{j+} [J mol^{-1}]	Anion	Δa_{j-} [J mol-1]
Li^+	7,120	Cl^-	-1050
Na^+	10,000	Br^-	-4,000
K^+	-4,190	CO_3^-	20,900
Mg^{2+}	31,000		
Ca^{2+}	16,800		

There does not seem to be a consensus in the literature on which method is the most appropriate. The viscosity method is convenient since it only requires knowledge of the diffusivity of the gas solute in pure water and the viscosities of pure water and the AES, which the later can evaluated with thermodynamic software such as CoolProp (Bell et al., 2014). However, Akita (1981) criticised the correlations with viscosity due to the spread of the empirical coefficient N (Figure 2.11) and suggested that the cause of the spread was that there was a variable not being accounted for in the correlation.

The salt concentration method, like the viscosity method, is relatively simple to implement since it requires knowledge only of the diffusivity in pure water, the salt concentration and one or two empirical coefficients depending upon which correlation is selected. Jamnongwong et al. (2010) proposed equation (2.40) as it is analogous to the Sechenov equation used when estimating gas

solubilities in AES (see section 2.3.2) and suggested that it could be extended to organic compounds. The difficulty of implementing the salt concentration methods comes from selecting the values for the empirical coefficients which seem to be poorly established and values of the empirical coefficients are limited in the composition of the salt as Jamnongwong et al. (2010) only presented values for sodium chloride solutions.

For the purposes of this work the Eyring theory method outlined by Akita (1981) is adopted due to the uncertainty of the values of the empirical coefficients in the other methods, its ease of implementation with different co-solutes, its intuitive correctness from its link to the kinetic theory of gases and becuase it predicted the diffusivities in AES within 13% of the observed values of Ratcliff and Holdcroft (1963).

2.3.2 Gas solubility

The behaviour of gas solubility in water still presents a significant technical challenge for HACs because, by Henry's law, the solubility of gases is proportional to the partial pressure of the gas as shown in Figure 2.12(Geng and Duan, 2010).

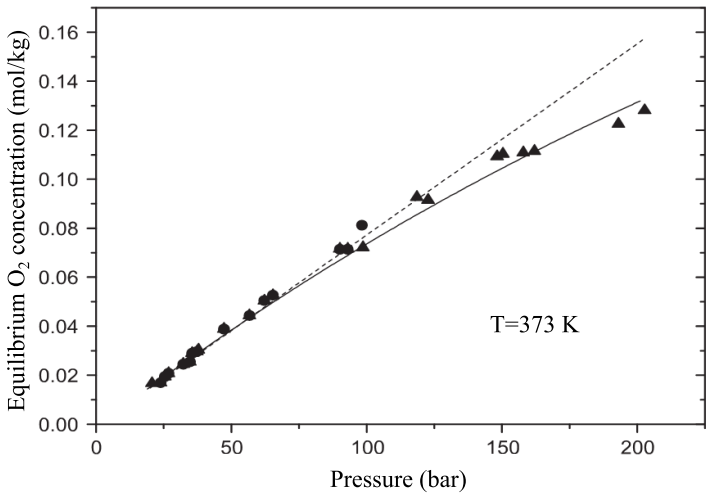


Figure 2.12 Equilibrium oxygen solubility vs pressure modified from Geng and Duan (2010)

When the water in a HAC is recirculated through a pump as proposed by Millar (2014), the equilibrium solubility behaviour of gases can be manipulated by increasing the water temperature and/or adding a co-solute (Figures 2.13 and 2.14).

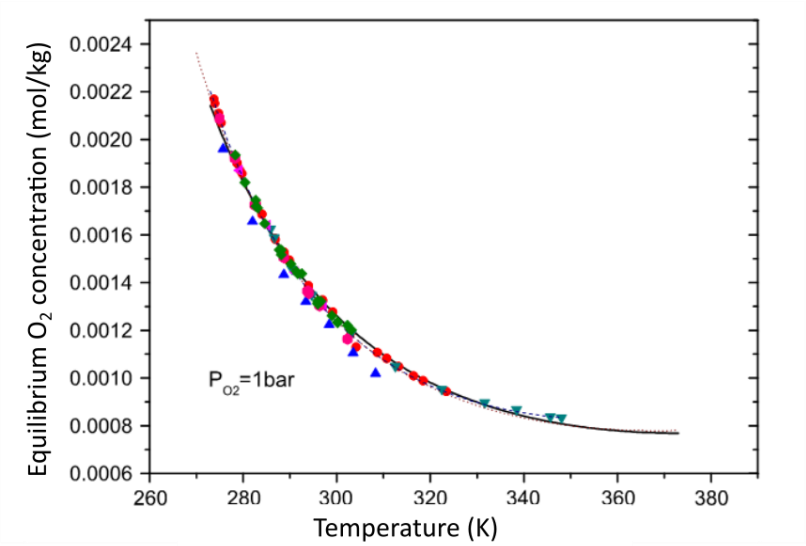


Figure 2.13 Equilibrium oxygen solubility vs temperature modified from Geng and Duan (2010)

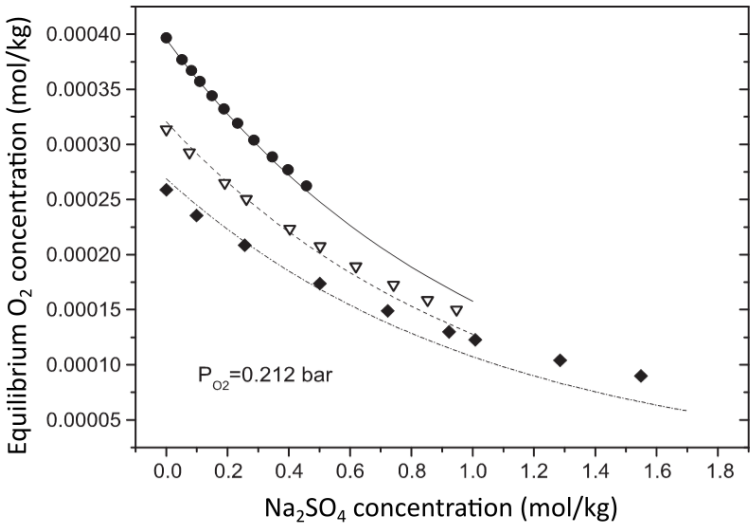


Figure 2.14 Equilibrium oxygen concentration vs co-solute concentration modified from Geng and Duan (2010)

When evaluating the gas absorption rate, based on two-film theory, it is assumed that the gas solute concentration at the gas-liquid interface is constantly at equilibrium concentration, that is,

that the resistance to molecular diffusion at the gas-water interface (bubble surface) is nil. Therefore, the partial pressure of the species in the gas phase at the gas-liquid interface ($p_{j,2}$ in Figure 2.10) can be converted to a concentration in the liquid phase ($C_{j,2}$ in Figure 2.10) through Henry's law (Sander, 2015):

$$C_{j,eq} = H_j^{cp} \chi_j P \tag{2.46}$$

Here $C_{j,eq}$ is the equilibrium concentration of species j in the liquid phase in mol/m^3, H_j^{cp} is the Henry's solubility constant defined by concentration and pressure in mol /m^3 Pa, χ_j is the mole fraction of species j in the gas phase and P is the total pressure of the gas phase.

To accurately reflect the gas solubility behaviour with temperature (Figure 2.13), the Henry's solubility constant is corrected with the van 't Hoff equation (Sander, 2015):

$$H_j^{cp}(T) = H_{j,0}^{cp} \cdot \exp\left[\frac{-\Delta \check{h}_{sol}}{\mathcal{R}}(T^{-1} - T_0^{-1})\right] \tag{2.47}$$

Here $H_{j,0}^{cp}$ is the Henry's solubility constant at the reference temperature T_0 in mol/m^3·Pa, T_0 is the reference temperature of 298.15 K, T is the temperature at which the Henry's constant is to be evaluated in K, $\Delta \check{h}_{sol}$ is the molar enthalpy of dissolution in J/mol and $\mathcal{R}$ is the universal gas constant in J/mol K.

Sander (2015) performed an exhaustive literature review on the Henry's constants and tabulated the values of $H_{j,0}^{cp}$ and $-\Delta \check{h}_{sol}/\mathcal{R}$ for numerous substances with water as a solvent. The values for the common gas species of atmospheric air are shown in Table 2.6.

Table 2.6 Values used in the model to evaluate the Henry's constant of a given species at temperatures other than 298.15K (Sander, 2015)

Species	$H_{j,0}^{cp}$ [mol/m^3 Pa]	$-\Delta\check{h}_{sol}/_{\mathcal{R}}$ [K]
Nitrogen (N2)	6.4x10^{-6}	1300
Oxygen (O2)	1.3x10^{-5}	1500
Argon (Ar)	1.4x10^{-5}	1500
Carbon dioxide (CO2)	3.4x10^{-4}	2400

Henry's law as shown in equation (2.46) is only applicable when the solutions are dilute (x_j < 0.03), when the gas solute is non-reactive with the solvent (e.g. water) and pressures up to 20 bar (Skogestad, 2008) or a riser depth of 200 m. For pressures exceeding 20 bar, the Henry's law equation needs to be modified to account for the pressure influence on the liquid phase (Skogestad, 2008). Zheng and Yapa (2002) adopted the so-called "modified Henry's law" presented by King (1969) to account for the nonlinearity (shown in Figure 2.15) which arises at pressures higher than 20 bar.

$$f_j = H_j^{xp} x_j \exp\left[\frac{P\check{v}_j}{\mathcal{R}T}\right] \tag{2.48}$$

Here f_j is the fugacity (thermodynamic partial pressure) of the gas in the gas phase in Pa, H_j^{xp} is the Henry's constant defined by liquid phase mole fraction and pressure in Pa, T is the water temperature in K, $\check{v}_j$ is the partial molar volume of the gas solute in the liquid phase in m^3/mol and $\mathcal{R}$ is the universal gas constant in J/mol·K. Andrade (2013) reviewed several solubility models and recommended the model from Diamond and Akinfiev (2003) for the prediction of the solubility of carbon dioxide at pressures from up to 60 bar. Historically, HACs maximum operating pressure was 9.28 bar (Millar, 2014) which is well within the limits of the simplified Henry's law. For this reason the simplified Henry's law has been adopted for the purposes of this

work but the modified Henry's law (King, 1969) and the solubility model from Diamond and Akinfiev (2003) are potential options for future analyses at pressures exceeding 20 bar.

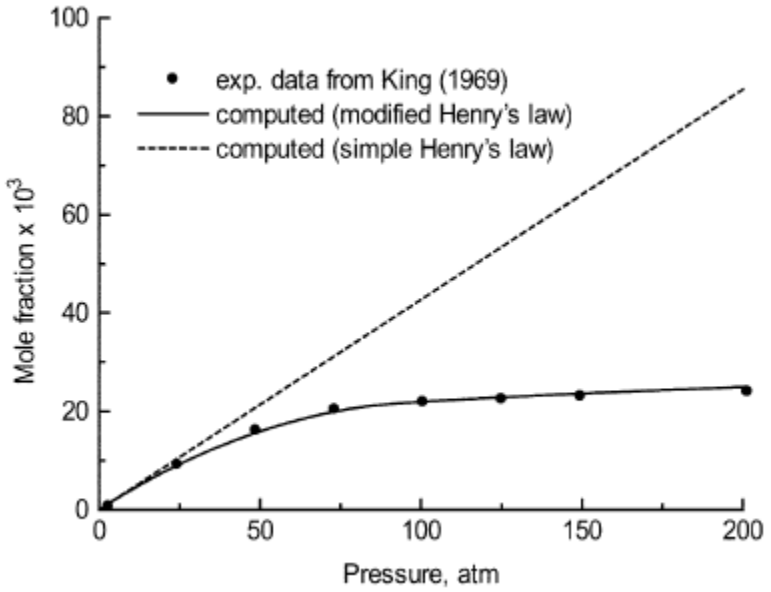


Figure 2.15 Equilibrium solubility of CO_2 in water with increasing pressure at 40°C (Zheng and Yapa, 2002)

In the case of carbon dioxide, dissolved CO_2 does react with water to produce carbonic acid. Normally Henry's law does not apply to solutes that react chemically with the solvent but the kinetics of carbonic acid formation is relatively slow and the amount of dissolved carbon dioxide that reacts to form carbonic acid at equilibrium is less than one percent (Brinkman et al., 1933; Higbie, 1935; Millar, 2014). Therefore it is assumed that the impact of the reaction between carbon dioxide and water can be neglected and that Henry's law still applies for carbon dioxide.

Like the mass diffusivity of gases in liquids, the solubility of gases is affected by the presence of dissolved salts in the water (Figure 2.14) and the Henry's constant needs to be adjusted accordingly. In contrast to the mass diffusivity, there is a method commonly used in the literature to adjust the Henry's constant due to dissolved salts in the water: the Sechenov equation (Sander, 2015; Schumpe, 1993):

$$\log\left(\frac{C_{j,eq,0}}{C_{j,eq}}\right) = \log\left(\frac{H_{j,0}^{cp}}{H_j^{cp}}\right) = k_S C_{cs} \tag{2.49}$$

Here $C_{j,eq,0}$ is the equilibrium concentration in pure water in mol/m^3, $C_{j,eq}$ is the equilibrium concentration in the AES in mol/m^3, $H_{j,0}^{cp}$ is the Henry solubility constant in pure water in mol/m^3 Pa, H_j^{cp} is the Henry solubility constant in the AES in mol/m^3·Pa, C_{cs} is the concentration of the salt co-solute in mol/m^3 and k_S is the Sechenov constant.

Weisenberger and Schumpe (1996) extended the approach from Schumpe (1993) and Hermann et al. (1995) to evaluate the Sechenov constant to account for the temperature dependence on the Sechenov constant. The Sechenov constant is evaluated with the following equations:

$$k_S = \sum(\lambda_j + \lambda_g) \cdot I_j \tag{2.50}$$

$$\lambda_g = \lambda_{g,0} + \lambda_T(T - T_0) \tag{2.51}$$

Here λ_j is the ion specific parameter of ion j in m^3/mol, λ_g is the gas specific parameter in m^3/mol, n_j is the index of ion j in the chemical formula for the salt, λ_T is the gas specific parameter for the temperature effect, $\lambda_{g,0}$ is the gas specific parameter at the reference temperature T_0 in m^3/mol, T is the temperature at which the Sechenov constant is evaluated and T_0 is the reference temperature of 298.15 K.

The parameters required for the atmospheric gases and a selection of cations and anions are tabulated in tables 2.7 and 2.8.

Table 2.7 Gas specific parameters and gas specific temperature parameters for common atmospheric gases (Weisenberger and Schumpe, 1996)

Species	$\lambda_{g,0}$ [$m^3 mol^{-1}$]	λ_T [$m^3 mol^{-1} K^{-1}$]
Nitrogen (N2)	-1.00E-06	-6.05E-07
Oxygen (O2)	0	-3.34E-07
Argon (Ar)	5.70E-06	-4.85E-07
Carbon dioxide (CO2)	-1.72E-05	-3.38E-07

Table 2.8 Selected ion specific parameters to evaluate Sechenov constant (Weisenberger and Schumpe, 1996)

Cation	λ_j [$m^3 mol^{-1}$]	Anion	λ_{j-} [$m^3 mol^{-1}$]
Li^+	7.540E-05	Cl^-	3.180E-05
Na^+	1.143E-04	Br^-	2.690E-05
K^+	9.220E-05	CO_3^-	1.423E-04
Mg^{+2}	1.694E-04		
Ca^{+2}	1.762E-04		

2.3.3 Psychrometry

The psychrometry of atmospheric air influences the gas absorption rate in a HAC by introducing another species to the gas mixture in the form of water vapour. This, in turn, reduces the partial pressures of the other gases in comparison to the air if it were assumed to be completely dry. Under certain conditions, HACs will produce compressed air that is dryer than the atmospheric

air (Chen and Rice, 1982). Since psychrometry impacts the driving force for mass transfer, it should be accounted for when modelling the air absorption rate.

The standard psychrometric equations model the mixture of dry atmospheric gases and water vapour as ideal gases where Dalton's law of partial pressures applies (McPherson, 2009).

$$\gamma = \frac{n_{H2O}}{n_a} = \frac{p_{H2O}}{p_a} \tag{2.52}$$

$$\gamma' = \frac{M_{H2O}}{M_a} \cdot \gamma \tag{2.53}$$

$$p_a = P - p_{H2O} \tag{2.54}$$

$$p_{H2O} = \phi \cdot p_{sat(T),H2O} \tag{2.55}$$

Here γ is the molar absolute humidity in mol H_2O / mol dry air, γ' is the absolute humidity in kg H_2O / kg dry air, M_{H2O} is the molar mass of water in kg/mol, M_a is the molar mass of the dry air mixture in kg/mol, p_a is the partial pressure of the dry air mixture in Pa, p_{H2O} is the partial pressure of water vapour in Pa, P is the total pressure of the gas mixture in Pa, $p_{sat(T),H2O}$ is the saturation vapour pressure of water at the temperature T in Pa and ϕ is the relative humidity of the mixture in Pa/Pa.

Typically the molar mass of dry air is taken to be the standard value of 0.02896 kg/mol but this value is based upon the standard dry composition of atmospheric air and in the case of the air absorption of the HAC the composition of the dry air mixture is changing based upon the air absorption history in the downcomer shaft. Therefore, the molar mass of the dry air mixture is based upon the local dry air composition (Chen and Rice, 1983):

$$M_a = \sum \chi_j' \cdot M_j \tag{2.56}$$

Here χ_j' is the dry mole fraction of species j in mol j/mol dry air and M_j is the molar mass of species j in kg/mol.

While the psychrometric equations were evaluated in terms of atmospheric air, the basis of the formulation is treating air and water vapour as a mixture of ideal gases. Therefore, the psychrometric equations are valid for gas mixtures other than air, provided that the dry gas composition is known.

2.4 Gas-liquid separators

Gas-liquid separators have two main classifications: i) gravity separators and ii) cyclonic separators. Gravity separators rely on the different densities of the gas and liquid phases so that the gas will separate from the liquid phase through buoyancy while cyclonic separators rely on the centrifugal forces generated by swirling the two-phase flow at high tangential velocities. Cyclonic separators tend to be more compact than gravity separators due to the accelerated separation from the cyclonic action (Mantilla et al., 1999).

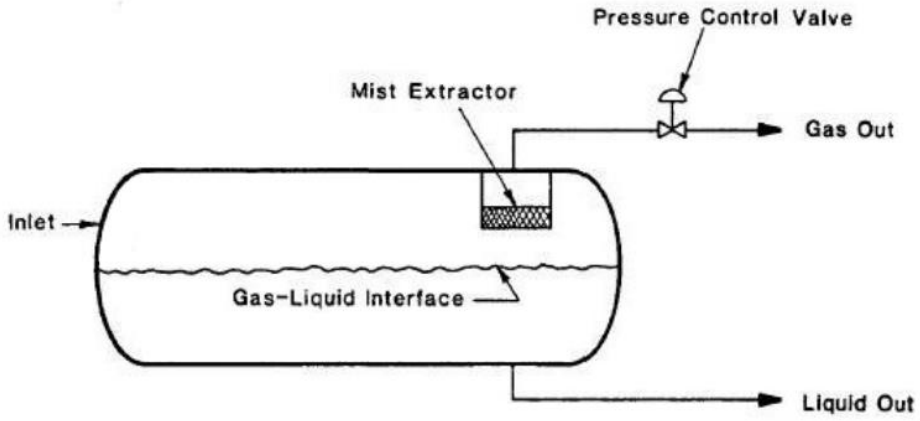


Figure 2.16 Schematic of a gravity separator used in the oil and gas industry (modified from Arnold and Stewart, 2008)

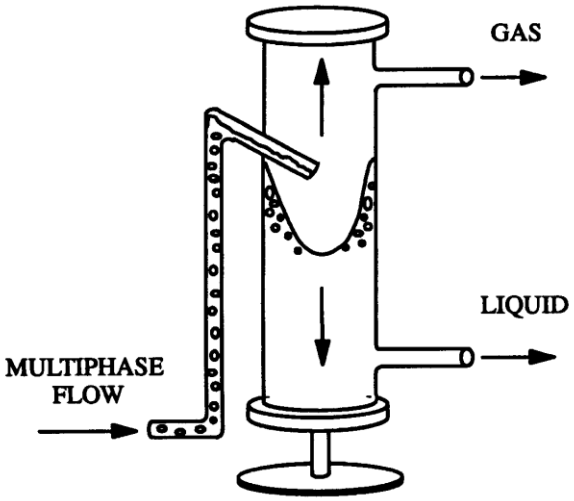


Figure 2.17 Schematic of a gas-liquid cyclone separator (Mantilla et al., 1999)

Whether gravity or cyclonic, the gas-liquid separator has two defining operational parameters: i) pressure drop and ii) separation efficiency (Kurokawa and Ohtaki, 1995). The pressure drop for gas-liquid separators is typically defined as the change in pressure from the inlet to the liquid outlet (Kouba et al., 1995). The separation efficiency of a gas-liquid separator is typically

defined as the ratio of the gas volume (or mass) flow rate at the gas outlet to the inlet (Kurokawa and Ohtaki, 1995). While there are some mechanistic models in the literature to evaluate these parameters (Kouba et al., 1995), typically they are evaluated through the use of computational fluid dynamic (CFD) codes (Laleh et al., 2012).

Based upon CFD simulations, Laleh (2010) shows the behaviour of the separation efficiency of gas dominated gas-liquid gravity separators in the oil and gas industry.

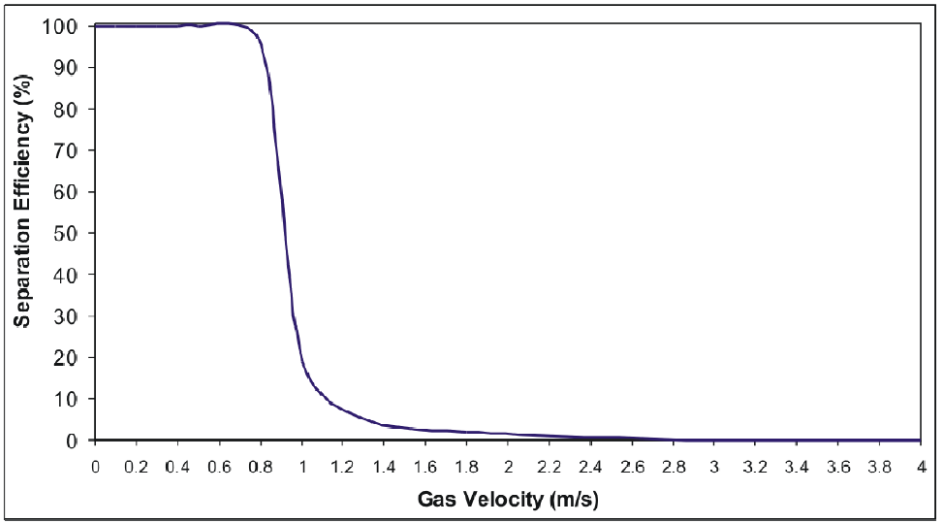


Figure 2.18 Separation efficiency vs inlet gas velocity (Laleh, 2010)

The incipient velocity or the velocity at which liquid carry over begins predicted by the CFD was found to match experimental data reasonably well except for two cases (Separator B and C at 70 kPa) as shown in the following table:

Table 2.9 Comparison of incipient velocities predicted by CFD and experimental results (Laleh, 2010)

Separator	Operating Pressure [kPa]	Incipient velocity [m/s]	
		Simulated	Experimental
A	70	1.6	-
	700	1.0	1.0 - 1.7
	2760	0.6	0.4 - 0.8
B	70	*1.2*	2.2 - 2.6
	700	0.8	0.4 - 1.4
	2760	0.5	0.1 - 0.5
C	70	*1.2*	2.2 - 2.6
	700	0.6	0.4 - 1.4
	2760	0.5	0.1 - 0.5
D	70	1.5	1.3 - 2.0
	700	1.0	0.7 - 1.7
	2760	1.0	0.4 - 0.8

In the context of the HAC the likelihood of liquid carry over (liquid exiting the gas outlet) is small at the typical mass flow rate ratios of liquid to gas at the two-phase inlet. Liquid carry over typically occurs in gas dominated gas-liquid separators (see Figure 2.18). For the gas-liquid separator in a HAC, the greater concern is gas carry under (gas exiting the liquid outlet).

Data in the literature for pressure drop across a gravity separator is not available because the gravity separator literature come from the oil and gas sector where the pressure drop is not a primary concern (Hutchison, 2016). Analyzing data from Zhao et al. (2004) shows the behaviour of pressure drop across a gas-solid cyclone separator with volumetric flow rate entering the separator. The data shows that the pressure drop is proportional to the square of the volumetric flow rate entering the separator inlet, of constant cross-sectional area, which can be represented with the following equation:

$$\Delta P = R \cdot \dot{V}_{in}^2 \tag{2.57}$$

Here ΔP is the pressure drop across the separator, $\dot{V}_{in}$ is the volumetric flow rate entering the separator and R is the resistance of the separator akin to the Atkinson resistance (McPherson, 2009) used in mine ventilation. The data from Zhao et al. (2004) could be reproduced with a value of 4.11×10^4 for R.

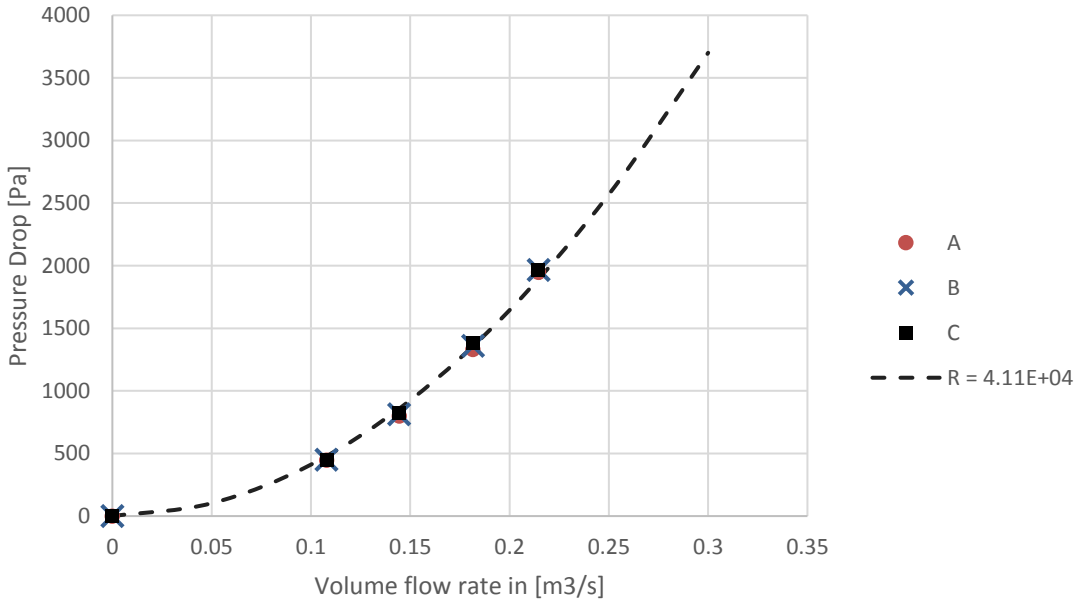


Figure 2.19 Pressure drop vs volumetric flow rate entering the model A, B and C cyclone separators from Zhao et al. (2004) and the line of $P = 4.11 \times 10^4 \cdot \dot{V}^2$

Kurokawa and Ohtaki (1995) examined the separation efficiency of a gas liquid cyclone separator with different volumetric flow rate ratios entering the separator, β. The results of which are shown in the Figure 2.20:

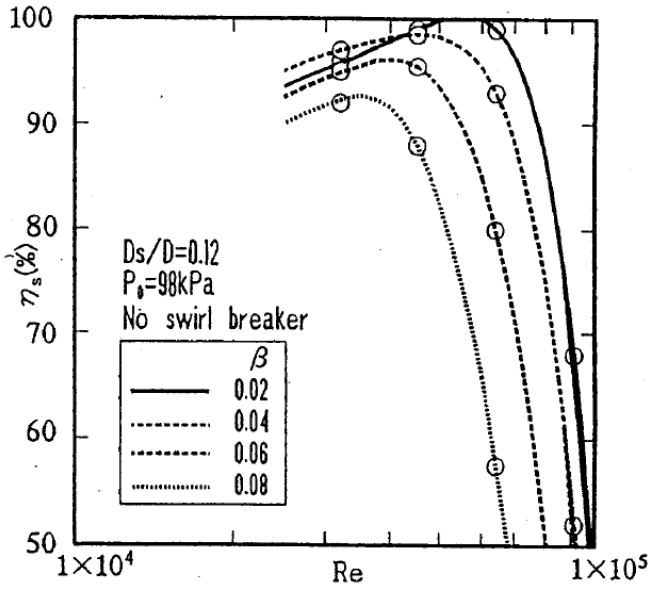


Figure 2.20 Separation efficiency vs Reynolds number in the cyclone with an aspect ratio (length to diameter) of 18 from Kurokawa and Ohtaki (1995)

Kurokawa and Ohtaki (1995) defined the Reynolds number with of the average axial velocity in the cyclone, the cyclone diameter and the kinematic viscosity of the fluids. A similar behaviour in cylindrical cyclones is observed as in gravity separators where at a certain velocity there is a large drop in separation efficiency of the separator.

2.5 Summary

In this chapter, the principal components required to effect the necessary theoretical upgrade to the formulation for HAC downcomer and riser hydrodynamics in Millar (2014) have been reviewed in detail. Where possible, when there have been formulation options to consider, the best option to assume in the HAC context has been adopted. The review has encompassed i) the possible two phase flow regimes, ii) bubble size, shape and size distribution which is important for determining the interfacial area for gas transfer, iii) hydrodynamic drag and iv) the

fundamental theory available detailing the rate of gas absorption or evolution across a gas liquid interface. In the next chapter, Chapter 3, the hydrodynamic formulation itself is detailed before a synthesis of the material in this chapter and the hydrodynamic formulation is enacted in Chapter 4.

Chapter 3

3. HAC modeling methodology

Applying the principles in the literature review, a model, shown schematically in Figure 3.1, was developed to simulate the HAC compression process in the downcomer using MATLAB. This work is principally focused on predicting the compressed air yield reduction from the gas absorption in the downcomer flow and is described in detail in Chapter 4 but this chapter presents a methodology for modelling the HAC process as a whole.

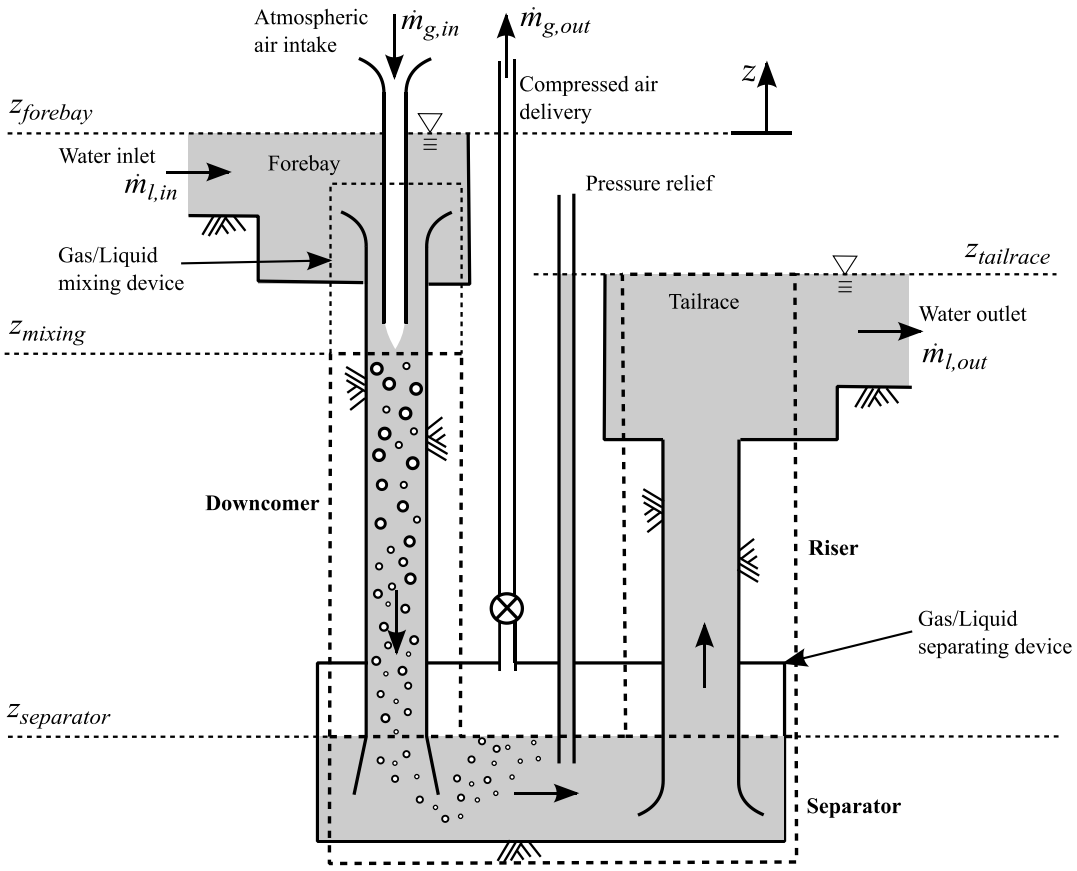


Figure 3.1 Schematic of the HAC model showing the different modelling domains in bold and key model inputs (Young et al., 2016)

In order to simulate the HAC a few key design parameters must be known (Figure 3.1): the water level elevations in the forebay, $z_{forebay}$, the tailrace, $z_{tailrace}$, and the separator, $z_{separator}$, the elevation of the point in the downcomer flow where the two phases are completely mixed, z_{mixing}, the mass flow rate of the liquid phase entering the system, $\dot{m}_{l,in}$, the liquid temperature at the water inlet, atmospheric pressure, and the diameters and absolute roughness of the downcomer and riser shafts. Equipped with the knowledge of these parameters one can solve for the mass flow rate of gas that is inducted into the system, $\dot{m}_{g,in}$.

The amount of air that is inducted by a HAC is the amount which balances the system such that the pressure at the forebay and tailrace water levels, $z_{forebay}$ and $z_{tailrace}$ in Figure 3.1 respectively, is atmospheric resulting from the flow losses through the process. The losses arise from friction, drag, mixing and separation irreversibilities and work done compressing the gas. The magnitude of these losses, including the "loss" due to gas compression, all depend on the magnitude of the mass flow rate of gas admitted to the system. Consequently, a complex implicit relationship for the inducted gas mass flow rate, $\dot{m}_{g,in}$, is established in this balance. Solving the hydrodynamics of the HAC thus amounts to building in all the necessary physics and chemistry to model the processes with sufficient accuracy and then solving for the value of $\dot{m}_{g,in}$ that satisfies the pressure boundary conditions. The basic hydrodynamic formulation, as well as the theoretical upgrades of Chapter 2, lead to a wildly non-linear system that requires solution using iterative methods.

In order to determine the losses through the system the mass flow rate of gas inducted by the HAC is initialised by averaging the reported mass flow rate ratios of water to air from the historical installations found in Millar (2014). Then the HAC is divided into different domains:

mixing, downcomer, separator and riser. The mixing domain evaluates the conditions at the inlet of the downcomer, defined by z_{mixing} in Figure 3.1, from atmospheric conditions. The downcomer model characterises the two phase bubbly flow of the downcomer shaft. The separator model evaluates the conditions at the riser inlet from the conditions at the downcomer outlet, and the riser evaluates the conditions at the tailrace tank surface. In general, the model overall, where the individual domain models are coupled, is one dimensional and any fluid properties required (e.g., density and viscosity) are evaluated using either REFPROP (Lemmon et al., 2013) or CoolProp (Bell et al., 2014) Each domain model is described in more detail in the following sections.

3.1 Mixing

Since the analysis of the air-water mixing process in a HAC is currently under investigation by Hutchison (2016), the mixing process in this work was modeled relatively simply. The primary function of the mixing domain is to take in the design parameters and atmospheric conditions and evaluate the state variables (i.e, pressure, temperature, molar flow rate of atmospheric gas species in the gas phase, molar flow rate of atmospheric gas species in the liquid phase, gas slip velocity, and liquid velocity) and bubble flux at the downcomer inlet.

In the mixing process, there must be a region where the pressure is below atmospheric to induct air. The downcomer inlet is then defined as the point where the two phase flow is fully mixed and the pressure has recovered to be equal to atmospheric pressure as shown in Appendix D. Consequently, the atmospheric pressure and the so-called “mixing elevation” (z_{mixing} in Figure 3.1) are required as input and design parameters respectively.

Regardless of the environmental temperature of the gas phase entering the HAC, through the mixing process the temperature of the gas at the inlet of the downcomer becomes that of the liquid phase and becomes saturated with water vapour (Chen and Rice, 1983). Using the standard atmospheric mole fractions of dry atmospheric air from Table 3.1, the dry mole fractions of the gas phase at the downcomer inlet are initialised.

Table 3.1 Standard mole fractions of the major gases in dry atmospheric air (Williams, 2016)

Species	Standard dry atmospheric mole fraction, χ' [mol / mol]
Nitrogen (N_2)	0.7808
Oxygen (O_2)	0.2095
Argon (Ar)	0.0093
Carbon Dioxide (CO_2)	0.0004

Then assuming that the gas phase is an ideal gas mixture and is at one hundred percent relative humidity the molar absolute humidity is evaluated using equations (2.52).The dry mole fractions of the gas species can then be renormalized to include water vapour in the gas mixture with the following equation:

$$\chi_j = \frac{\chi'_j}{(1+\gamma)} \tag{3.1}$$

Here χ_j is the mole fraction of species *j* including water vapour, hence called the "humid mole fraction" of species *j* and χ_j' is mole fraction of species *j* excluding water vapour, hence called the "dry mole fraction" of species *j*. To initialise the gas solute concentration in the bulk liquid

phase it is assumed that the water entering the HAC is at equilibrium conditions at the atmospheric conditions using Henry's Law with equation (2.46). Equipped with the mole fractions of gas species in the gas phase and gas solute concentrations in the liquid phase, the mass flow rates of the gas and liquid phase are decomposed into their individual component molar flow rates with the following equations:

$$\dot{n}_{j,g} = \frac{\dot{m}_g}{M_g} \cdot \chi_j \tag{3.2}$$

$$\dot{n}_{j,l} = \frac{\dot{m}_l}{\rho_l} \cdot C_{j,B} \tag{3.3}$$

$$\dot{n}_{j,T} = \dot{n}_{j,g} + \dot{n}_{j,l} \tag{3.4}$$

This is necessary so that the quantity of each gas species in both phases can be tracked through the downcomer process.

From Rice's (1976) experiments it was found that a value of 0.244 m/s as the initial slip velocity of the gas phase at the downcomer inlet produced the best agreement with his model and experimental values. In this work the methodology outlined by Millar (2014) is adopted where the Sauter mean bubble diameter at the downcomer inlet is evaluated using the correlations from Akita and Yoshida (1974) or Wilkinson et al. (1994) and Rice's initial slip velocity value is used as an initial estimate to solve for the slip velocity which balances the drag and buoyancy forces on a bubble iteratively.

The average liquid velocity is determined with the following equation (Bhagwat and Ghajar, 2012):

$$U_l = \frac{U_{sl}}{(1-\alpha)} \tag{3.5}$$

Here U_l is the average liquid velocity in m/s, U_{sl} is the superficial liquid velocity in m/s and α is the volume fraction of gas as determined using equation (2.27).

Another requirement of the downcomer model is to know the bubble flux, or the number of bubbles per second, entering the downcomer which is determined using the following equations:

$$B = \frac{\dot{m}_g}{m_b} \tag{3.6}$$

$$m_b = \rho_g \cdot \frac{\pi}{6} \cdot d_b^3 \tag{3.7}$$

Here B is the bubble flux in 1/s, m_b is the mass of an average bubble in the bubbly flow in kg and d_b is the Sauter mean bubble diameter of the bubbly flow in m. Now having solved for the state variables and the bubble flux at the downcomer inlet, the downcomer model can now be used to simulate the hydrodynamic, psychrometric, and solubility kinetic processes in the gas compression process.

3.2 Downcomer

As shown in section 2.2.1, the two phase flow in the downcomer changes from a slug or transitional flow at the inlet to a bubbly flow during the transit in the downcomer but the approach taken for the sake of simplicity is that downcomer is modelled as a bubbly flow throughout. The bubbly flow in the downcomer is considered to be steady, one dimensional and is assumed to have a constant bubble flux. While the bubble flux is constant, the mass flow rate of gas in the downcomer varies due to gas solubility and hence the bubble mass varies in the

downcomer. The full formulation of the downcomer flow is described in detail in Chapter 4 but the general solution procedure is outlined herein.

The downcomer is simulated discretely by dividing the downcomer into a number of segments. The values of the state variables at the inlet and outlet of the first segment are initialised using the values of the state variables at the downcomer inlet evaluated from the mixing domain. The values at the outlet are regarded as initial estimate of pressure, temperature, liquid velocity, and gas slip velocity. These are then refined using the Newton-Raphson algorithm with numerical partial derivatives (Keffer, 1999) while solving the one-dimensional equations of conservation of energy, momentum, mass and terminal slip velocity, also assuming initially no mass is transferred between phases and there are constant fluid properties in the segment. This solution for the downcomer provides initial conditions for the numerical solution of all of the state variables including the molar flow rate of each species in the gas phase at the segment outlet. The conservation of energy, momentum, mass, and terminal slip velocity equations are augmented by species conservation equations in this final solution. The values of the state variables at the segment outlet are then assigned to the inlet of the following segment and the process is repeated until the state variables at the outlet of the downcomer have been solved for.

The Newton-Raphson algorithm flexes the values of the state variables to evaluate the gradient of the conservation equations' residuals (Chapter 4) and iteratively refines the values of the state variables. The solution is considered converged when the root-mean-square (RMS) of the difference of the state variables from one iteration to the next is below a defined tolerance of $1x10^{-8}$ (which was, nevertheless, above the double precision of the computations). During this procedure the total molar flow rate (in liquid and gas) of each species is kept constant and when

the algorithm flexes the molar flow rate of a given species in the gas phase, this change is reflected in the molar flow rate in the liquid phase, as well as the mass flow rate of each phase.

Once each segment of the downcomer has been solved, the RMS of the residuals in each segment are evaluated to judge if the error in the overall solution is acceptable. If so, the power losses due to wall friction and bubble drag are summed and stored to evaluate the system efficiency subsequently. The values of the state variables at the downcomer outlet are then passed to the separator to evaluate the conditions at the riser inlet.

3.3 Separator

The key properties of the gas-liquid separator, shown schematically in Figure 3.2, in the context of predicting the compressed air yield and solving for the mass flow rate of air inducted by a HAC are: i) the pressure drop across the separator from the two phase inlet to the water outlet and ii) the separation efficiency of the separator.

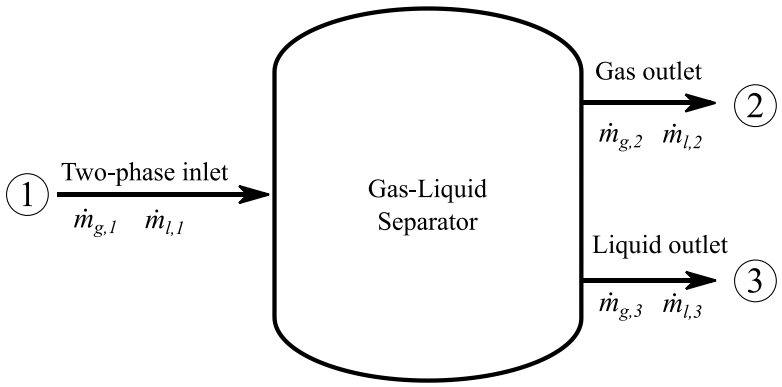


Figure 3.2 Schematic of the gas liquid separator in a HAC

The separation efficiency of the gas-liquid separator is defined with the following equation:

$$\eta_{sep} = \frac{\dot{m}_{g,2}}{\dot{m}_{g,1}} \tag{3.8}$$

In the ideal case with 100% separation efficiency, $\dot{m}_{g,3}$ and $\dot{m}_{l,2}$ should both equal zero. The approach taken to model the gas-liquid separator's pressure drop is to assign it an Atkinson resistance akin to what is used in mine ventilation. Then the pressure drop across the separator can be evaluated with the Atkinson equation (McPherson, 2009):

$$P_1 - P_3 = R \cdot \dot{V}_1^2 \tag{3.9}$$

Here P_1 is the pressure at the two phase inlet in Pa, P_3 is the pressure at the water outlet, R is the Atkinson's resistance of the separator, and $\dot{V}_1$ is the total volumetric flow rate entering the separator at the two-phase inlet in m^3/s.. It is assumed that the exit and entry losses from the downcomer and riser shafts respectively are accounted for within the separator resistance. In this work, both the resistance and separation efficiency are considered design parameters and are assigned before the analysis.

Equipped with the resistance and separation efficiency of the separator and the diameter of the riser shaft, the state variables (i.e., pressure, temperature, average liquid velocity, gas slip velocity and molar flow rate of each species in both phases) and bubble flux at the riser inlet is determined. The pressure at the riser inlet is evaluated by modifying equation (3.9) to the following equation:

$$P_{R,in} = P_{D,out} - R \cdot \dot{V}_{D,out}^2 \tag{3.10}$$

Here $P_{R,in}$ is the pressure at the riser inlet in Pa, $P_{D,out}$ is the pressure at the downcomer outlet in Pa and $\dot{V}_{D,out}$ is the volumetric flow rate of liquid at the downcomer outlet in m^3/s. It is assumed that the difference in temperature across the separator is negligible.

The liquid velocity and the gas slip velocity at the riser inlet are evaluated in the same manner as for the downcomer inlet. However, the Sauter mean bubble diameter must be evaluated differently since the flow conditions at the riser inlet are outside of the intended use of the correlations from Akita and Yoshida (1974) and Wilkinson et al. (1994) which were developed empirically for bubble columns to produce the maximum stable bubble size from a given superficial gas velocity being sparged into a column with a given diameter. The Sauter mean bubble diameter at the riser inlet depends on the bubble size distribution entering the gas-liquid separator and the gas-liquid separator's separation efficiency.

The bubbles entering the gas-liquid separator do not have a uniform size but have a log-normal distribution and can be represented by the Rosin-Rammler distribution using equation (2.12). The downcomer model, however, outputs the Sauter mean bubble diameter which is not equal to the Rosin-Rammler mean. Akita and Yoshida (1974) demonstrate how to evaluate the Sauter mean diameter from a histogram of bubble diameter observations which can be simulated using the Rosin-Rammler distribution taking the spread parameter to be 2.5 (Johansen et al., 2000). The Rosin-Rammler mean bubble diameter of the underflow, or the gas that exits at the water outlet, is determined by evaluating the bubble diameter where the fraction of bubbles entering the separator that is greater than the Rosin-Rammler mean of the under flow is equal to the separation efficiency of the separator. Then the distribution of the underflow is simulated assuming the same spread parameter as the distribution at the inlet (see Figure 3.3). Finally, the Sauter mean bubble diameter of the underflow is evaluated and the gas slip velocity and the bubble flux at the riser inlet are then solved for using the same process as at the downcomer inlet.

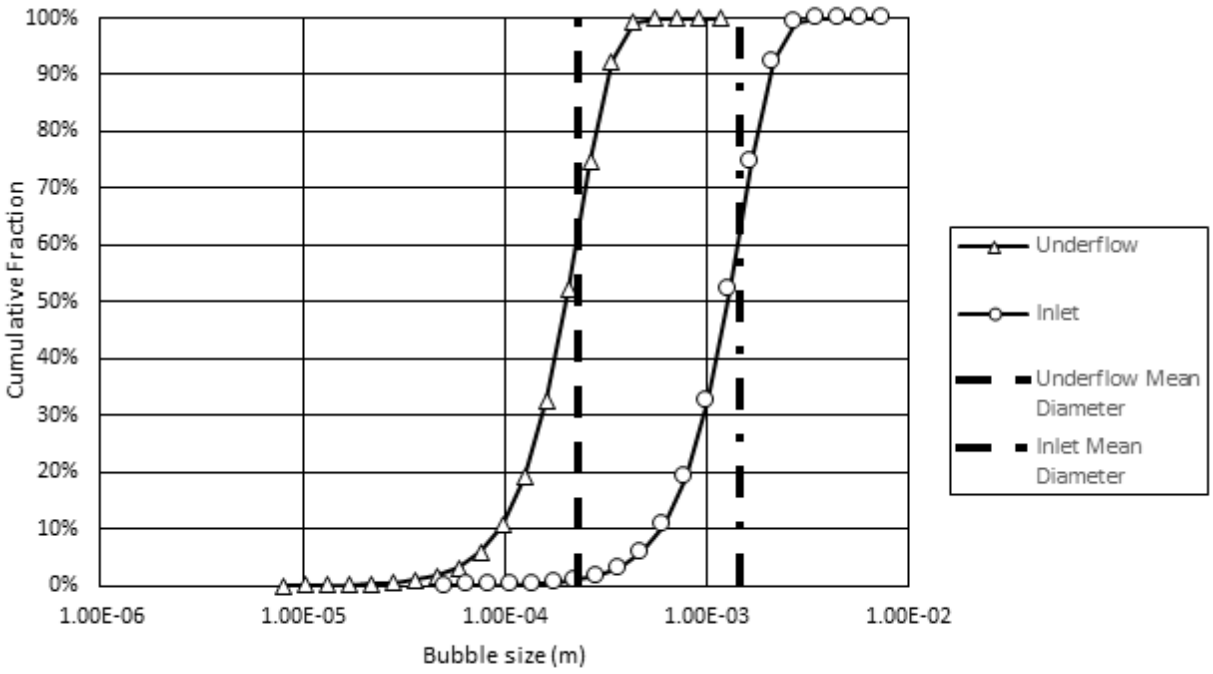


Figure 3.3 Cumulative fraction of the two-phase inlet and underflow of a gas-liquid separator with a separation efficiency of 99% for the example shown in Appendix G

Since there is significantly less interfacial are in the separator than in the downcomer, it is assumed that there is no mass transfer in the gas-liquid separator and, therefore, the molar flow rate of the species in the liquid phase at the riser inlet is considered to be equivalent to those at the downcomer outlet and the molar flow rates of the species in the gas phase at the riser inlet are evaluated simply by using the following equation for each species:

$$\dot{n}_{j,g,R,in} = \left(1 - \eta_{sep}\right) \cdot \dot{n}_{j,g,D,out} \tag{3.11}$$

where η_{sep} is the separation efficiency of the gas-liquid separator.

When modelling the gas-liquid separator in the HAC, the ideal case is to assume a separator with a resistance of zero and a separation efficiency of one hundred percent. In other words, all of the gas entering the separator reaches the gas outlet and the pressure at the water outlet is the same

as the pressure at the two phase inlet. This being the case, the flow at the riser inlet is a single phase flow with of liquid only. The gas slip velocity, bubble flux and the molar flow rates of species in the gas phase do not need to be evaluated and the equations for the remaining state variables simplify to the single phase. It is important to note that in the ideal case the compressed air yield of the HAC is only affected by the gas absorbed into the liquid phase in the downcomer and the separator does not impact the efficiency of the HAC or the amount of air inducted by the HAC. With the state variables at the riser inlet now solved for, the last modelling domain of the HAC is used to evaluate the state variables at the tailrace elevation.

3.4 Riser

How the riser shaft is modelled depends upon the design separation efficiency of the gas-liquid separator. If the separation efficiency is one hundred percent, the riser shaft is modelled as a single phase incompressible liquid flow. This simplification allows the state of the liquid (i.e., pressure, temperature and velocity) to be evaluated explicitly using the steady flow energy equation (Cengel and Cimbala, 2010). Because the riser shaft has a constant cross-section and is assumed to be incompressible, the riser flow has a constant velocity and density. The riser flow can then be expressed with the following equation:

$$\frac{P_{R,in}}{\rho_R} + \frac{U_R^2}{2} + gz_{R,in} = \frac{P_{R,out}}{\rho_R} + \frac{U_R^2}{2} + gz_{R,out} + e_{mech,loss} \tag{3.12}$$

Here $e_{mech,\ loss}$ is specific energy loss due to frictional flow losses and the exit loss from the flow entering the reservoir in the tailrace area which can be evaluated with the following equation:

$$e_{mech,loss} = \left(f\frac{L}{d_R} + K_{L,exit}\right)\frac{U_R^2}{2} \tag{3.13}$$

Here f is the friction factor evaluated from the Colebrook equation and $K_{L,\,exit}$ is the exit loss coefficient which is equal to 2 for fully developed laminar flow and 1.05 for fully developed turbulent flow (Cengel and Cimbala, 2010).

If the separation efficiency is less than one hundred percent, then the riser flow is a bubbly flow and the downcomer model can be repurposed to evaluate the riser flow in the riser shaft. As a consequence of the change in flow direction of bubbly flow in the downcomer to the bubbly flow in the riser is that flowing liquid around the bubbles no longer resists the buoyancy of the bubbles and the bubble drag in the riser flow promotes flow. Hence, while considered a loss in the downcomer, the bubble drag is considered an energy gain in the riser. How this is accounted for in the downcomer model is simply through the sign convention of the gas slip velocity such that the sign of the slip velocity in the riser flow is negative.

Whether the riser is modelled as a single or two phase flow, at present bubbles generated from exsolved gas are neglected in the HAC cycle model since the constant bubble flux assumption of the downcomer model prevents bubbles from being spontaneously generated in the flow. Subsequently, this will be different. The downcomer process model can be applied to the riser flow to permit the exsolution of gas to be modelled, as the air absorption process in reverse. New bubbles are not generated because bubble nucleation, division and coalescence are not represented in the model. However underpassing existing bubbles with a distribution like that shown in Figure 3.3 will increase in mass in the riser as gas is exsolved. This represents the capacity in the model herein to be applied to the riser process of the HAC cycle in due course, when a complete experimental program merits this. The formulation of downcomer model is presented in full in the following chapter.

Chapter 4

4. Simultaneous modeling of hydrodynamic, psychrometric and solubility kinetic processes

The downcomer flow is modelled as a one dimensional two phase bubbly flow. The approach outlined by Rice (1976) is adopted where the flow is discretized into a series of segments, the pressure and temperature of both phases are considered equal at every section of the flow and the buoyancy of the gas phase is accounted for with a slip velocity. These are reasonable assumptions which have the added benefit of reducing the amount of unknowns that need to be solved and simplifies some of the equations necessary to solve for them.

The parameters required to define the state of the fluids at each section are the pressure, temperature, liquid velocity, gas slip velocity, and molar flow rate of each species in the gas mixture. For each segment of the flow the equations defining the conservation of energy, conservation of momentum, conservation of mass, the terminal slip velocity condition and the conservation of each species of the gas mixture are solved simultaneously using the Newton-Raphson procedure (Keffer, 1999) to solve for the state of the fluids at the outlet of each segment. The problem is in its simplest form when no mass transfers are permitted where it has 4 unknowns to solve for with 4 equations. When mass transfers are permitted the size of the problem depends on the number of components in the gas mixture. For example, if air is approximated with nitrogen and oxygen only, the problem has 6 unknowns and 6 equations and similarly, if air is treated as a mixture of nitrogen, oxygen, argon and carbon dioxide, the problem has 8 unknowns and 8 equations. In general the formulation is such that if the

absorption of a gas mixture with N components the size of the problem has $N+4$ unknowns which requires $N+4$ equations.

The following sections describe how these equations are formulated. The formulation adopts the approach of Chen and Rice (1983) when possible to model the solubility kinetics and psychrometrics so that the modelling results could be compared. However, likely due to limitations of journal paper length, some details are missing in the publication outlined by Rice (1976) (e.g., how the air absorption model was coupled with the hydrodynamic model and how the volume fraction of gas was evaluated). The gaps in the formulation were filled using appropriate sources found in the literature.

4.1 Conservation of energy

To model the hydrodynamics, solubility kinetics, and psychrometrics simultaneously of the downcomer flow, there are two primary considerations when formulating the conservation of energy for a given segment of the downcomer: i) the flow is two phase and ii) mass must be permitted to transfer between phases. The control volume for a given segment in the downcomer flow is as shown in Figure 4.1.

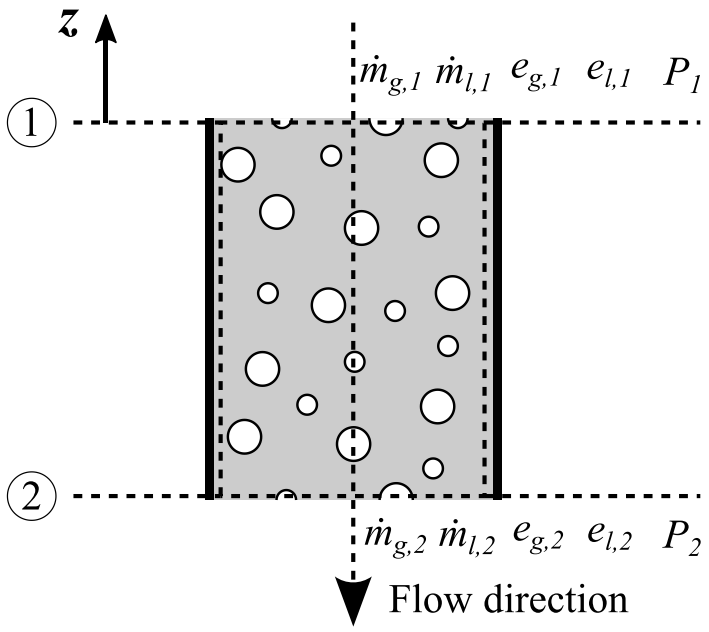


Figure 4.1 Control volume for a given segment

The conservation of energy equation for a given segment is formulated such that the only work input to the control volume is from the pressure forces at the inlet and outlet and the flow is assumed to be adiabatic, such that the conservation of energy is expressed with the following equation, taking energy entering the control volume to be positive:

$$\dot{m}_{l,1}\left(\frac{P_1}{\rho_{l,1}}+e_{l,1}\right)+\dot{m}_{g,1}\left(\frac{P_1}{\rho_{g,1}}+e_{g,1}\right)=\dot{m}_{l,2}\left(\frac{P_2}{\rho_{l,2}}+e_{l,2}\right)+\dot{m}_{g,2}\left(\frac{P_2}{\rho_{g,2}}+e_{g,2}\right) \quad (4.1)$$

$$e_{l,k}=\frac{U_{l,k}^2}{2}+gz_k+u_{l,k} \quad (4.2)$$

$$e_{g,k}=\frac{\left(U_{l,k}-U_{s,k}\right)^2}{2}+gz_k+u_{g,k} \quad (4.3)$$

Here position *1* is the inlet of a given segment, position *2* is the outlet and *k* is either position *1* or *2*. It is important to note that due to the influence of solubility kinetics $\dot{m}_{g,1}$ is not equal to $\dot{m}_{g,2}$ (because of gas mass lost from gaseous phase due to dissolution) and similarly $\dot{m}_{l,1}$ is not equal to

$\dot{m}_{l,2}$ (because of gas mass gained due to dissolution), the mechanical losses in the control volume are accounted for in the change of internal energy of the fluids and the elevations z_1 and z_2 are in metres above datum or "mAD" with the datum being the elevation of the forebay water level ($z_{tailrace}$ as shown in Figure 3.1).

The conservation of energy as expressed in equation (4.1) neglects the heat (or enthalpy) of solution for the gases absorbed in a segment and the heat (or enthalpy) of condensation of any water vapour in the segment. According to Perry and Green (1999) the conditions where the heat effects must be accounted for are i) the gas solute has an appreciable heat of solution and ii) a large amount of gas solute is absorbed in the liquid phase.

Since the main components of air are only slightly soluble in air, their heat of solutions are relatively small as compared to hydrochloric acid for which the heat effects in absorption must be accounted for (Perry and Green, 1999) as shown in Table 4.1.

Table 4.1 Molar enthalpies of solution for the main components of atmospheric air and hydrogen chloride

Species	Enthalpy of solution [kJ/mol]	Source
Nitrogen	13.21	Benson and Krause (1976)
Oxygen	14.67	Benson and Krause (1976)
Argon	14.35	Benson and Krause (1976)
Carbon dioxide	20.00	Duan and Sun (2003)
Hydrogen chloride	74.79	Vanderzee and Nutter (1963)

In a given segment the amount of gas that is absorbed in the liquid phase and the amount of water vapour that is condensed is low such that the heat of solution and heat of condensation is small relative to the other energy terms. This was confirmed with a simulation of the whole

downcomer in the Ragged Chutes HAC. The change of enthalpy due to the absorption of gases and condensation of water vapour was three orders of magnitude lower than the enthalpy of the water in the downcomer. Therefore, it is assumed that it is safe to neglect the heat of solution and heat of condensation.

4.2 Conservation of momentum

The conservation of momentum can be expressed using the steady one-dimensional linear momentum equation (Cengel and Cimbala, 2010):

$$\sum \boldsymbol{F} = \sum_{in} \dot{m}\boldsymbol{U} \cdot \hat{\boldsymbol{n}} + \sum_{out} \dot{m}\boldsymbol{U} \cdot \hat{\boldsymbol{n}} \tag{4.4}$$

Here $\hat{\boldsymbol{n}}$ is the outward facing normal vector of the control surface, $\boldsymbol{F}$ is a force vector and $\boldsymbol{U}$ is a velocity vector. Adopting the sign convention of flow direction positive results in the dot product of the momenta vectors and the outward facing normal of the inlet control surface always being negative. The formulation is also one-dimensional so that the velocities are the average velocity normal to the control surface and the forces are resolved into the components which are parallel to the flow direction. Therefore equation (4.4) now becomes:

$$\sum F_{\beta} = \sum_{out} \dot{m}U - \sum_{in} \dot{m}U \tag{4.5}$$

Since the downcomer flow is two phase, the momentum of both phases must be conserved and requires the momenta of both phases to be evaluated at the inlet and outlet of a given segment. The forces acting on the control volume of a downcomer segment are the pressure forces at the inlet and outlet, the pressure force on the wall, the force of fluid friction and the weights of both phases. The drag and buoyancy forces on the bubbles do not appear in the formulation because

even though they act on the control surfaces of the inlet and outlet, the terminal slip velocity condition applied in the formulation results in the drag and buoyancy forces being equal and opposite forces. When the flow direction is not vertically downward, only a component of the buoyancy force is in the direction of flow but that component is still negated by the drag force. The component of buoyancy which is not negated by drag would cause bubbles to be more concentrated in upper portion of the duct but how this affects the flow is neglected in the formulation. The forces and momenta acting on the control volume of a downcomer segment are shown in Figure 4.2

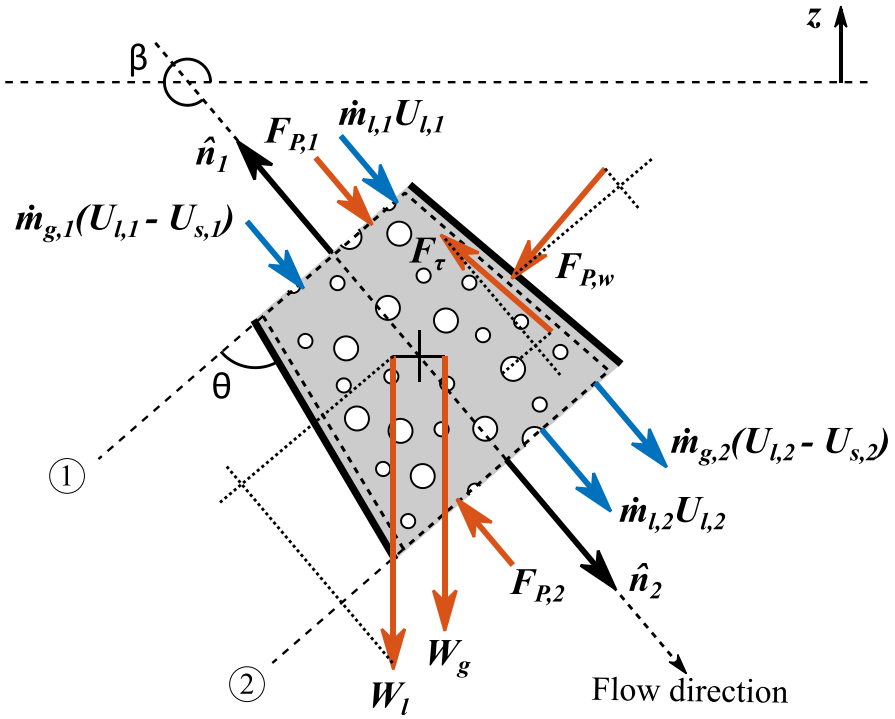


Figure 4.2 Conservation of momentum control volume showing the forces (red), momenta (blue), the outward facing normal vectors of the control surfaces, flow direction angle *β* and wall angle *θ*

The formulation is generalised such that bubbly flows with any flow direction in divergent, convergent or straight walled ducts can be accommodated by it. This is accomplished by treating the control volume as a conical frustum and two angles: i) the flow direction angle, β, and ii) the wall angle, θ. The flow direction angle is the angle measured counter clockwise positive from the right directed horizontal to the centre axis of the flow. The wall angle is measured from the inlet counter clockwise to the duct wall. Using these angles the components of the weight, friction and wall pressure forces which are in the direction of flow can be evaluated.

The magnitude of the fluid weight of the liquid and gas phases are evaluated with the following equations:

$$W_l = \frac{\rho_{l,1} + \rho_{l,2}}{2} \cdot g \cdot V_{seg} \cdot (1 - \alpha) \tag{4.6}$$

$$W_g = \frac{\rho_{g,1} + \rho_{g,2}}{2} \cdot g \cdot V_{seg} \cdot \alpha \tag{4.7}$$

$$V_{seg} = \frac{\pi L}{12}(d_1^2 + d_1 d_2 + d_2^2) \tag{4.8}$$

Here α is the volume fraction of gas in the segment, L is the length of the segment measured perpendicularly from the inlet to the outlet in m, d_1 is the hydraulic mean diameter of the inlet in m and d_2 is the hydraulic mean diameter of the outlet in m. This approach assumes that the densities of the liquid and gas phases vary linearly across the segment. The component of the liquid phase fluid weight that is acting in the direction of the flow is evaluated with the following equation:

$$W_{l,\beta} = -\sin\beta \cdot W_l \tag{4.9}$$

and similarly for the gas phase

$$W_{g,\beta} = -\sin\beta \cdot W_g \tag{4.10}$$

The magnitude of the friction force is evaluated using the following equation:

$$F_\tau = \tau \cdot \mathrm{A_w} \tag{4.11}$$

where:

$$\tau = \frac{1}{8} \cdot f \cdot \rho \cdot U^2 \tag{4.12}$$

$$A_w = \frac{\pi l}{2}(d_1 + d_2) \tag{4.13}$$

$$l = \sqrt{L^2 + \frac{1}{4}(d_1 - d_2)^2} \tag{4.14}$$

and l is the slanted length of the wall in the downcomer segment in m and A_w is the surface area of the wall in m^2 and τ is the shear stress on the wall in Pa.

When evaluating the shear stress on the wall, the friction factor is determined through the use of the Haaland approximation (4.15) as an initial guess which is then refined using the Colebrook equation (4.16) (Cengel and Cimbala, 2010).

$$\frac{1}{\sqrt{f}} = -1.8\log\left[\frac{6.9}{N_{Re}} + \left(\frac{\varepsilon/d}{3.7}\right)^{1.11}\right] \tag{4.15}$$

$$\frac{1}{\sqrt{f}} = -2.0\log\left(\frac{\varepsilon/d}{3.7} + \frac{2.51}{N_{Re}\sqrt{f}}\right) \tag{4.16}$$

$$N_{Re} = \frac{\rho U d}{\mu} \tag{4.17}$$

Here N_{Re} is the Reynolds number of the flow, ρ is a characteristic density of the flow in kg/m^3, U is the characteristic velocity in m/s, d is the characteristic of diameter in m, and μ is the characteristic viscosity in Pa s. Since the flow in the downcomer is two phase the selection of the characteristic parameters of the flow to evaluate the Reynolds number require careful selection. The model evaluates the parameters for the Reynolds number in two ways: i) using the liquid phase properties only or ii) using a mass flow weighted average of both phases. When using the liquid phase properties the characteristic velocity, density and viscosity are simply a linear average of the values at the inlet and outlet while the characteristic diameter is taken to be the diameter at the outlet. When the parameters are evaluated as mass flow weighted averages of both phases, the characteristic diameter is, again, taken as the outlet diameter while the characteristic density, velocity and viscosity are evaluated using the following equations:

$$\rho = \frac{\dot{m}_{l,1}}{\dot{m}_{T,1}} \cdot \left(\frac{\rho_{l,1} + \rho_{l,2}}{2}\right) + \frac{\dot{m}_{g,1}}{\dot{m}_{T,1}} \cdot \left(\frac{\rho_{g,1} + \rho_{g,2}}{2}\right) \tag{4.18}$$

$$U = \frac{\dot{m}_{T,1}}{\rho A_2} \tag{4.19}$$

$$\mu = \frac{\dot{m}_{l,1}}{\dot{m}_{T,1}} \cdot \left(\frac{\mu_{l,1} + \mu_{l,2}}{2}\right) + \frac{\dot{m}_{g,1}}{\dot{m}_{T,1}} \cdot \left(\frac{\mu_{g,1} + \mu_{g,2}}{2}\right) \tag{4.20}$$

Now that the shear stress on the wall and the magnitude of the fluid friction force are known, the component of the fluid friction force which is in the direction of the flow can be determined using the following equation:

$$F_{\tau,\beta} = -\sin\theta \cdot F_{\tau} \tag{4.21}$$

Since the fluid friction force is always resisting the flow, the component of fluid friction force in the flow direction is always negative.

The magnitude of the wall pressure force is determined using the following equation:

$$F_{P,w} = \frac{(P_1 + P_2)}{2} \cdot A_w \tag{4.22}$$

The component of the wall pressure force that is in the direction of the flow is then determined using the wall angle:

$$F_{P,w,\beta} = F_{P,w} \cos\theta \tag{4.23}$$

When the downcomer segment is a divergent duct (θ is less than 90°), $F_{P,w,\beta}$ is in the same direction of the flow, when the downcomer segment is a convergent duct (θ is greater than 90°), $F_{P,w,\beta}$ is in the opposite direction of the flow and when the downcomer segment is a straight duct (θ is equal to 90°), $F_{P,w,\beta}$ is zero. All of these conditions are accommodated in how the wall angle is defined and the properties of cosine give $F_{P,w,\beta}$ the appropriate sign.

The magnitudes of the pressure forces at the inlet and outlet are determined in the same manner as shown in the following equation:

$$F_{P,k} = P_k \cdot A_k \tag{4.24}$$

However, the pressure force at the outlet is always in the opposite direction of flow while the pressure force at the inlet is always in the same direction of the flow but both forces are perpendicular to their respective control surfaces. Thus the components of the forces in direction of flow are determined as shown in the following equations:

$$F_{P,1,\beta} = F_{P,1} \tag{4.25}$$

$$F_{P,2,\beta} = -F_{P,2} \tag{4.26}$$

Having resolved all the forces in the control volume (Figure 4.2) equation (4.5) now becomes

$$\sum F_{\beta} - \sum M = 0 \tag{4.27}$$

$$\sum F_{\beta} = W_{l,\beta} + W_{g,\beta} + F_{P,1,\beta} + F_{P,2,\beta} + F_{\tau,\beta} + F_{P,w,\beta} \tag{4.28}$$

$$\sum M = \dot{m}_{l,2}U_{l,2} + \dot{m}_{g,2}\left(U_{l,2} - U_{s,2}\right) - \dot{m}_{l,1}U_{l,1} - \dot{m}_{g,1}\left(U_{l,1} - U_{s,1}\right) \tag{4.29}$$

Equations (4.27) is then used as an additional constraint upon the Newton-Raphson solver to solve for the state variables at the outlet of the segment.

4.3 Conservation of mass and species

When modelling the solubility kinetics and psychrometrics in tandem with the hydrodynamics, the mass conservation equation must permit mass to transfer between phases to allow gases to dissolve into the water and for water vapour to condense and enter the liquid stream. Therefore the mass conservation is formulated such that the total mass flow rate of both phases is constant. The control volume of a downcomer segment is shown in Figure 4.3.

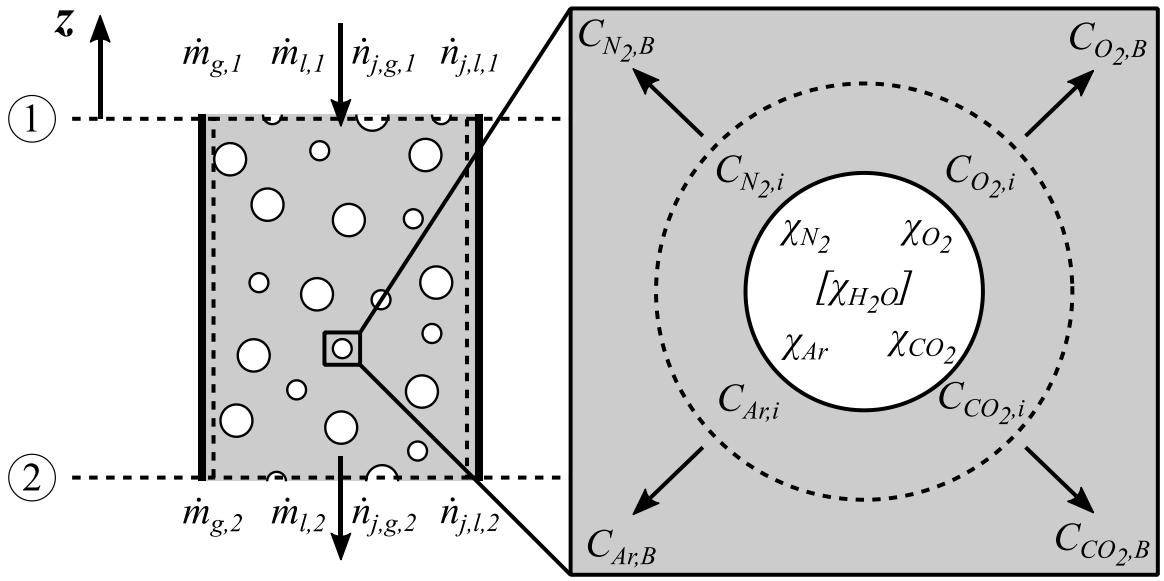


Figure 4.3 The control volume of a downcomer segment showing relevant parameters for the conservation of mass and species

The approach outlined by Rice (1976) of using the occupied areas of the phases was extended to include both phases and the mass conservation in a downcomer segment:

$$\dot{m}_{l,1} + \dot{m}_{g,1} = \rho_{l,2} U_{l,2} A_{l,2} + \rho_{g,2}\left(U_{l,2} - U_{s,2}\right) A_{g,2} \tag{4.30}$$

$$A_{l,2} = \frac{\dot{m}_{l,2}}{\rho_{l,2} U_{l,2}} \tag{4.31}$$

$$A_{g,2} = A_{D,2} - A_{l,2} \tag{4.32}$$

Here *1* refers to the inlet of a given downcomer segment and *2* refers to the outlet.

The modelling of the solubility kinetics requires that the conservation of mass be extended to each species in the gas mixture such that the mass of each species is conserved in the process. The conservation of species is formulated such that the change in the molar flow rate of a given

species in the gas phase is equal to the change in the molar flow rate of the species in the liquid phase:

$$\Delta\dot{n}_j = \dot{n}_{j,g,1} - \dot{n}_{j,g,2} = \dot{n}_{j,l,2} - \dot{n}_{j,l,1} \tag{4.33}$$

The sign convention taken for the conservation of species is mass transferring to the liquid phase is positive because this is the expected behaviour.

The gas absorption rate in the segment is assumed to be liquid phase dominated such that the resistance of a given species diffusing through the gas mixture to the gas-liquid interface is negligible and the limiting resistance is in the liquid phase. Hence the absorption rate of a given species, $\Delta\dot{n}_j$, is only a function of the concentration difference of the gas solute at the interface and the bulk liquid (Figure 4.3). Modifying the approach from Chen and Rice (1983) to be applied to a segment of the downcomer, the change in molar flow rate is evaluated with the following equations:

$$\Delta\dot{n}_j = K_j \cdot LMCD_j \cdot A_i \tag{4.34}$$

$$K_j = 2\sqrt{\frac{D_{j,l}}{\pi \cdot t_{e,avg}}} \tag{4.35}$$

$$LMCD_j = \frac{\left(C_{j,i,1} - C_{j,B,1}\right) - \left(C_{j,i,2} - C_{j,B,2}\right)}{\ln\left(C_{j,i,1} - C_{j,B,1}\right) - \ln\left(C_{j,i,2} - C_{j,B,2}\right)} \tag{4.36}$$

$$A_i = \frac{6 \cdot \alpha}{d_{b,avg}} \cdot V_{seg} \tag{4.37}$$

$$t_{e,avg} = \frac{1}{2}\left(\frac{U_{s,1}}{d_{b,1}} + \frac{U_{s,2}}{d_{b,2}}\right) \tag{4.38}$$

$$C_{j,i,k} = H_{j,k}^{cp} \cdot \chi_{j,k} \cdot P_k \tag{4.39}$$

$$C_{j,B,k} = \frac{\rho_{l,k}}{\dot{m}_{l,k}} \cdot \dot{n}_{j,l,k} \tag{4.40}$$

$$d_{b,k} = \left[\frac{6 \cdot \dot{m}_{g,k}}{\pi \cdot B \cdot \rho_{g,k}}\right]^{1/3} \tag{4.41}$$

Here $d_{b,avg}$ is the linear average of the Sauter mean bubble diameters at the inlet and outlet of the segment in m, B is the bubble flux which is constant through the downcomer in s^{-1}, K_j is the mass transfer coefficient as given by Higbie (1935), $LMCD_j$ is the log mean concentration difference of species j across a segment, A_i is the total interfacial area of each bubble resident in the segment, $D_{j,l}$ is the mass diffusivity of species j in the liquid phase in m^2/s as specified in section 2.3.1, $\chi_{j,k}$ and $\dot{n}_{j,l,k}$ are the local mole fraction of species j in the gas phase and the local molar flow rate of species j in the liquid phase, respectively, resulting from the air absorption history of the downcomer flow. The mole fraction, $\chi_{j,k}$ is the dry or humid mole fraction of species j depending if psychrometry is included in the analysis (see section 4.5).

The log mean concentration difference is adopted because it is commonly used when evaluating mass transfer (Subramanian, 2015) but if there is a point in the numerical procedure where log mean of the concentration differences would produce imaginary numbers, i.e., the concentration difference at the inlet has a different sign than the concentration difference at the outlet, a simple arithmetic average is used. This would occur if the direction of mass transfer changes in the segment. Since the molar flow rates of argon and carbon dioxide in the gas phase are small, the direction of mass transfer may change across the segment during the numerical procedure but since the interfacial concentration of gases is continually increasing in the downcomer process, it is only a numerical and not a physical behaviour. This could occur in the riser flow if the liquid phase is not at saturation at the riser inlet. Then at some point in the riser, the reduction of static

pressure would cause the direction of mass transfer to reverse and the gas would exsolve from solution.

During the numerical procedure, the Newton-Raphson algorithm will flex the molar flow rates of the dry species in the gas phase and based upon their values in a given iteration of the procedure the mass flow rates of both phases at the outlet must be determined to evaluate the residuals of the other conservation equations. As previously stated, whenever the algorithm flexes the molar flow rate of a species in the gas phase, the change is reflected in the liquid phase such that the total molar flow rate of a given species in the downcomer is kept constant. The mass flow rates at the outlet of the segment are adjusted using the following equations:

$$\Delta\dot{m}_{sol} = \sum \Delta\dot{n}_j \cdot M_j \tag{4.42}$$

$$\dot{m}_{g,2} = \dot{m}_{g,1} - \Delta\dot{m}_{sol} \tag{4.43}$$

$$\dot{m}_{l,2} = \dot{m}_{l,1} + \Delta\dot{m}_{sol} \tag{4.44}$$

When psychrometry is included in the model there is an additional mass transfer due to the condensation of water vapour and this is covered in section 4.5. Substituting equation (4.34) into (4.33) the conservation of species becomes:

$$\dot{n}_{j,g,1} - \dot{n}_{j,g,2} = K_j \cdot LMCD_j \cdot A_i \tag{4.45}$$

Equation (4.45) is equation used to constrain the solution of the numerical procedure for each dry component of the gas mixture included in the model.

4.4 Conservation of bubbles

The formulation for the bubbly flow of the downcomer assumes a constant bubble flux which means that no bubbles are created or destroyed in a segment. The bubble flux is required as an

input to the model and is determined using the procedure described in section 3.1. Consequently, any effect of bubble coalescence or bubble breakup are neglected and the only parameters that affect the size of an average bubble in the downcomer flow is the local mass flow rate of gas and the density of the gas mixture. When this is applied to the bubbly flow in the riser shaft, the constant bubble flux assumption will ignore the effects of bubbles being generated from gases evolving from solution but captures the effect by the air absorption process becoming a stripping process where dissolved gases are removed from the liquid phase and transfer to the gas phase.

The formulation also maintains a terminal slip velocity condition on the bubbles in the downcomer, i.e., the buoyancy and drag forces on a given bubble are balanced as shown in Figure 4.4.

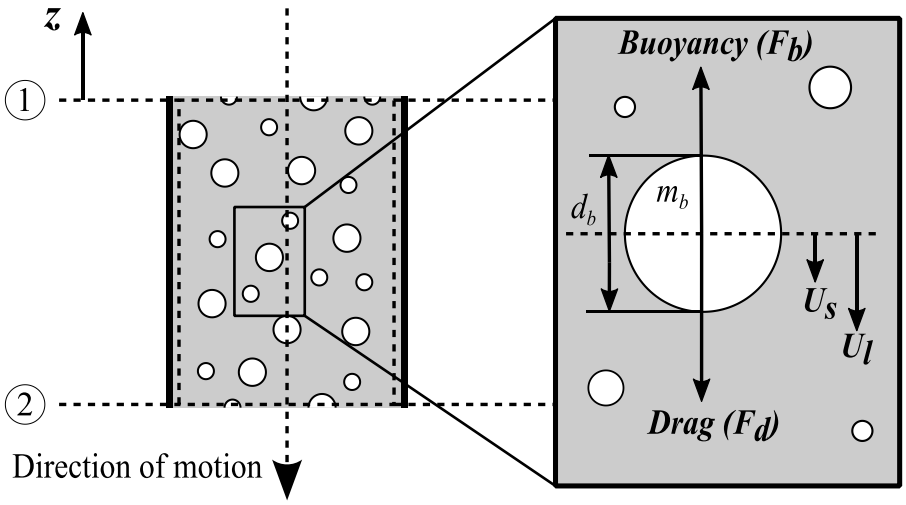


Figure 4.4 The force balance on a given bubble in a downcomer segment

The sign convention adopted for the gas slip velocity is flow direction positive. The direction of the gas slip velocity is determined by considering a reference frame moving at the speed of a bubble. With this reference frame in the downcomer flow, the liquid would be slipping past the

bubble since it is moving at a faster velocity and the slip velocity would be downward in the direction of the flow. The slip velocity of the gas can then be expressed through the following equation:

$$U_{l,k} = U_{g,k} + U_{s,k} \tag{4.46}$$

Here $U_{g,k}$ is the absolute velocity of the gas phase.

The applying the terminal velocity constraint, a gas slip velocity is found which satisfies the following equation modified from Chen & Rice (1983):

$$U_{s,k} = \sqrt{\frac{4}{3} \cdot \frac{d_{b,k} \cdot g \cdot (\rho_{l,k} - \rho_{g,k})}{c_d \cdot \rho_{l,k}}} \tag{4.47}$$

Here $d_{b,k}$ is the Sauter mean diameter of the bubbles as determined by equation (4.41), the drag coefficient is determined using equations (2.24) to (2.26) and the volume fraction of gas is determined using equation (2.27). This is an implicit relationship which must be solved simultaneously with the other conservation equations.

4.5 The effect of psychrometry

The model was developed such that the hydrodynamics and solubility kinetics could be simulated neglecting the psychrometrics of atmospheric air for comparison purposes. When psychrometry is omitted from the analysis, the only mass transfer between phases is due to solubility and the gas mixture is considered to be dry atmospheric air.

When psychrometry is included in the analysis, there is an additional mass transfer that must be evaluated. As the pressure increases through the downcomer flow, any water vapour held in the atmospheric air that is inducted will tend to condense and join the liquid phase. Using the

provisional values of pressure, temperature, dry gas composition and assuming the gas is constantly at 100% relative humidity in the downcomer, the absolute humidity of the gas mixture is evaluated using equation (2.53) and the mass transfer rate due to condensation is evaluated explicitly.

$$\dot{m}_{a,1} = \dot{m}_{g,1} \cdot (1 - \omega_{H2O}) \tag{4.48}$$

$$\omega_{H2O} = \frac{M_{H2O}}{M_g} \cdot \chi_{H2O} \tag{4.49}$$

$$M_g = \sum \chi_j \cdot M_j \tag{4.50}$$

$$\dot{m}_{a,2} = \dot{m}_{a,1} - \Delta\dot{m}_{sol} \tag{4.51}$$

$$\Delta\dot{m}_{H2O} = \gamma_1' \cdot \dot{m}_{a,1} - \gamma_2' \cdot \dot{m}_{a,2} \tag{4.52}$$

Here positions *1* and *2* are the inlet and outlet respectively, $\dot{m}_{g,k}$ is the total mass flow rate of the gas phase at position *k* in kg/s, $\dot{m}_{a,k}$ is the mass flow rate of dry air at position *k* in kg/s, ω_{H2O} is the mass fraction of water vapour in the gas phase, M_g is the molar mass of the humid gas mixture in kg/mol, $\Delta\dot{m}_{sol}$ is the change in mass transfer rate due to solubility in kg/s, γ' is the absolute humidity of the gas mixture in kg/kg and $\Delta\dot{m}_{H2O}$ is the mass transfer rate due to condensation of water. Now the total mass transfer in the segment is evaluated using the following equation

$$\Delta\dot{m} = \Delta\dot{m}_{sol} + \Delta\dot{m}_{H2O} \tag{4.54}$$

and equations (4.43) and (4.44) now become:

$$\dot{m}_{g,2} = \dot{m}_{g,1} - \Delta\dot{m} \tag{4.54}$$

$$\dot{m}_{l,2} = \dot{m}_{l,1} + \Delta\dot{m} \tag{4.55}$$

Additionally, in order to satisfy the total mass conservation the properties of the gas phase must be in terms of humid air. Therefore the mass flow rate of the total gas phase is in kg of humid air per second, the density is kg of humid air per m^3, the internal energy of the gas phase is in J per kg of humid air, etc. This requires the mole fractions of each component of the gas mixture to be known, including water, and so the fluid properties are evaluated with CoolProp or REFPROP thermodynamic software. When evaluating the air absorption due to gas solubility the mole fraction χ_j in equation (4.39) is the humid mole fraction, i.e., moles of species *j* per mole of humid air.

4.6 The effect of a co-solute

One of the interventions proposed by Millar (2014) to control the solubility behaviour in HACs was the use of a co-solute. One of the main objectives of this work was to include in the formulation the means of simulating the effect that a co-solute would have on the hydrodynamics and the solubility kinetics of the downcomer bubbly flow. The parameters that are affected when including a co-solute in the downcomer model are the fluid properties of the liquid phase, the Henry's constants used to evaluate the interfacial concentration and the mass diffusivities of the species in the liquid phase.

As previously stated, the fluid properties in the HAC model are evaluated through the use of the thermodynamic software package CoolProp (Bell et al., 2014). When simulating the downcomer without a co-solute present, the thermodynamic fluid properties of the liquid phase are taken to be that of pure water. Because the mole fractions of the gas solutes are on the on the order of $1x10^{-5}$ mol/mol, it is assumed that any effect that the gas solutes have on the thermodynamic properties can be neglected. This changes, however, when considering the amount of co-solute

required. As an illustrative example, Millar (2014) states that the co-solute concentration that would reduce the solubility of the gases by half is of the order of 1 mol/kg H_2O. This is approximately a mole fraction of 0.02 which is approaching the limits of the diffuse characterisation and is three orders of magnitude greater than the gas solute. Therefore it is not reasonable to assume that the co-solute's effect on the thermodynamic fluid properties of the liquid phase can be neglected. Fortunately, CoolProp has means of determining the thermodynamic properties of a selection of AES. The AES available from CoolProp and the limits of each are shown in Table 4.2**.**

Table 4.2 Aqueous electrolyte solutions available in CoolProp and limits on temperature, *T*, and mass fraction, *ω*, for each.

AES	T_{min} [°C]	T_{max} [°C]	T_{base} [K]	ω_{min}	ω_{max}
Lithium-bromide	-0.15	226.85	386.5	0	0.75
Calcium Chloride	-100	40	280.68	0	0.3
Potassium Carbonate	-100	40	284.39	0	0.4
Lithium Chloride	-100	40	274.64	0	0.24
Magnesium Chloride	-100	40	282.47	0	0.3
Sodium Chloride	-100	40	285.77	0	0.23

Here ω_{min} and ω_{max} are the minimum and maximum mass fraction (ratio of the mass of the co-solute to the mass of the solution) of the co-solute allowed permited by CoolProp.

Millar (2014) proposed using sodium sulphate as a co-solute in the HAC but this is not available in CoolProp. The density, viscosity and internal energy of of the AES in Table 4.2 were evaluated at 101,325 Pa, temperatures ranging from 5°C to 40°C and mass fractions of 5% and 15% to determine how much variance was exhibited in each of the fluids based upon co-solute

composition. The results are shown in figures 4.5, 4.6 and 4.7. It is important to note that the viscosity information for aqueous lithium bromide solutions are not available in CoolProp.

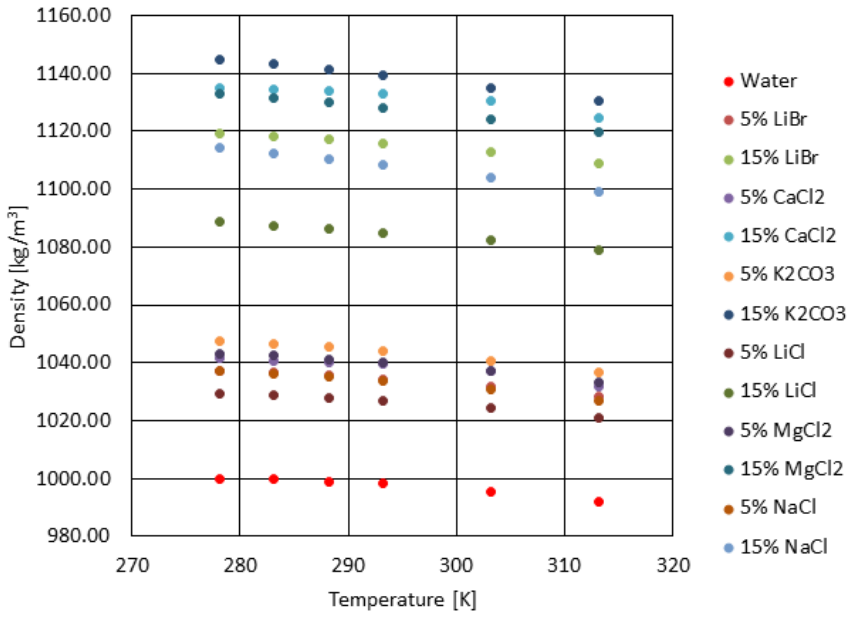


Figure 4.5 Variance of AES density [kg/m^3] by temperature and co-solute composition and concentration

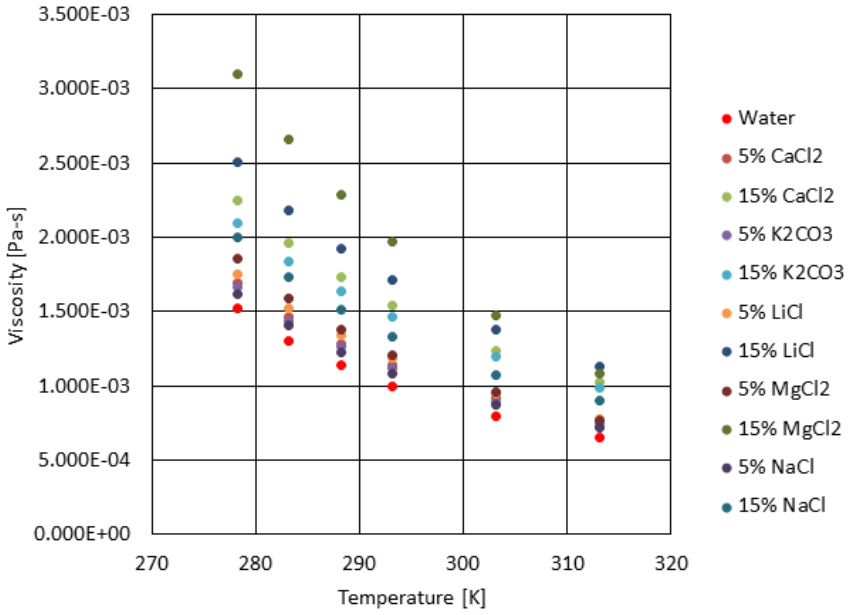


Figure 4.6 Variance of AES viscosity [Pa·s] by temperature and co-solute composition and concentration

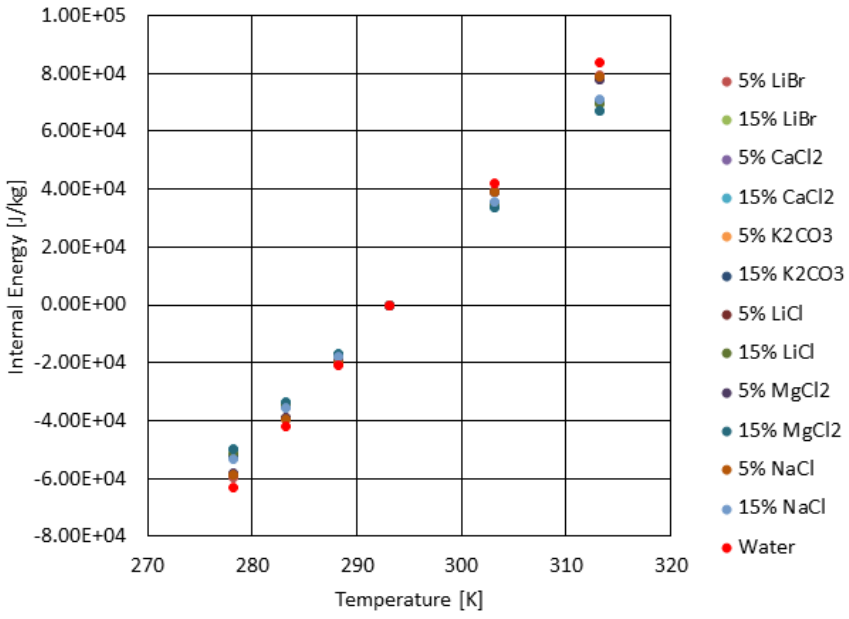


Figure 4.7 Variance of AES internal energy [J/kg] by temperature and co-solute composition and concentration

As expected, there is a large spread with the AES density with co-solute composition and concentration. There is a significant spread in the viscosities but they appear to converge with increasing temperature. The AES internal energies are very similar to each other which should be expected since the solution is mostly water. This suggests that provided the density is properly accounted for, sodium sulphate could be modeled in the HAC provided an appropriate co-solute analogue is found for the viscosity and internal energy.

The Henry's constants for a given species are required to evaluate the local interfacial concentration in the downcomer flow. The Henry's constants evaluated through equation (2.46) are used to obtain the equilibrium concentration of a gas species in pure water. However, for most salts there is a so-called "salting-out" effect where gas solubility decreases with increasing salt concentration (Weisenberger and Schumpe, 1996). The salting-out effect is captured through the use of the Sechenov equation (2.49) and the approach outlined by Weisenberger and Schumpe (1996) is adopted to modify the Henry's constants.

In addition to modifying the solubility of gases, the presence of a co-solute also inhibits the molecular diffusion of the dissolved gas molecules in the liquid phase (Ratcliff and Holdcroft, 1963). The model accounts for this by modifying the mass diffusivity by integrating the approach outlined by Akita (1981) with equations (2.41) to (2.45). This modified mass diffusivity is then applied in equation (4.35) so that with the modified interfacial concentrations, the effect on the solubility kinetics is fully captured.

The additional information that is required when modeling the effect of a co-solute is summarised in Table 4.3.

Table 4.3 Summary of information required to model the effect of a co-solute

Information required	Where it is applied	Source
Sechenov constant	Interfacial concentration, equation (4.39)	Weisenberger and Schumpe (1996)
Modified mass diffusivity	Mass transfer coefficient, equation (4.35)	Akita (1981)
Liquid phase density, viscosity and internal energy	Conservation of energy, conservation of momentum, conservation of mass, terminal slip velocity	CoolProp (Bell et al., 2014)

4.7 Formulation summary

The following equations summarises the equations that are solved by the Newton-Raphson algorithm

$$\dot{m}_{l,1}\left(\frac{P_1}{\rho_{l,1}}+e_{l,1}\right)+\dot{m}_{g,1}\left(\frac{P_1}{\rho_{g,1}}+e_{g,1}\right)-\dot{m}_{l,2}\left(\frac{P_2}{\rho_{l,2}}+e_{l,2}\right)-\dot{m}_{g,2}\left(\frac{P_2}{\rho_{g,2}}+e_{g,2}\right)=\zeta_1 \tag{4.56}$$

$$\sum F_\beta-\sum M=\zeta_2 \tag{4.57}$$

$$\dot{m}_{l,1}+\dot{m}_{g,1}-\rho_{l,2}U_{l,2}A_{l,2}-\rho_{g,2}\left(U_{l,2}-U_{s,2}\right)A_{g,2}=\zeta_3 \tag{4.58}$$

$$U_{s,k}-\sqrt{\frac{4}{3}\cdot\frac{d_{b,k}\cdot g\cdot\left(\rho_{l,k}-\rho_{g,k}\right)}{c_d\cdot\rho_{l,k}}}=\zeta_4 \tag{4.59}$$

$$\dot{n}_{j,g,1}-\dot{n}_{j,g,2}-K_j\cdot LMCD_j\cdot A_i=\ \zeta_{4+j} \tag{4.60}$$

where ζ_k is the residual of the equation k and equation (4.60) is repeated for each species in the gas mixture.

Chapter 5

5. Predicting the reduction of compressed air yield due to solubility

The formulation described in the previous chapter was implemented in MATLAB utilising the object oriented programming features such that a succinct or verbose solution to the downcomer flow could be attained depending upon which properties were queried. The model was used to simulate the downcomer decoupled from the entire HAC system in multiple scenarios which flex the parameters that affect the air absorption in the downcomer shaft to ensure that the model is behaving correctly. The parameters that were flexed were the mixture components of atmospheric air, depth of the downcomer outlet, intake liquid temperature, mass flow rate ratio, and co-solute concentration.

5.1 Gas mixture

To simplify the analysis, Chen and Rice (1983) approximated dry atmospheric air as a mixture of 79% nitrogen and 21% oxygen by mole. The formulation presented in Chapter 4 was developed such that there is flexibility in the species and number of the components in the gas mixture so the approach of Chen and Rice (1983) could be extended to include argon and carbon dioxide in the approximation of atmospheric air with capacity to include many more.

To test the influence of the initial dry gas composition on the mole fraction of oxygen at the downcomer outlet, four different gas mixtures were inducted to the HAC as shown in Table 5.1.

Mixtures I to III are different approximations of atmospheric air and mixture IV is an approximation of flue gas from a coal fired power plant.

Table 5.1 Dry gas mixtures used in scenario testing

Mixture	Initial dry mole fraction, χ' [mol / mol]			
	N_2	O_2	Ar	CO_2
I	0.79	0.21	0.00	0.00
II	0.78	0.21	0.01	0.00
III	0.7808	0.2095	0.0093	0.0004
IV	0.75	0.05	0.01	0.19

The downcomer of Ragged Chutes was used for the model testing so that the mole fractions of oxygen at the downcomer outlet predicted by the model could be compared with the measured value reported by E.B.W. (1910) and McNair and Koenig (1911) and the modeling results reported by Chen and Rice (1983). Using the geometric parameters from Schulze (1954), the Ragged Chutes downcomer was simulated using the following input parameters:

Table 5.2 Downcomer model inputs for simulating the air absorption in the Ragged Chutes HAC downcomer

Input parameter	Value	Units
Inlet water mass flow rate	29,690	kg/s
Inlet air mass flow rate	18.2	kg/s
Inlet pressure	101,325	Pa
Inlet temperature	294.15	K
Shaft diameter	2.591	m
Shaft length	100.83	m
Flow direction	-90	°
Absolute roughness	$1x10^{-3}$	m
Properties used to evaluate wall shear stress	Liquid phase only	
No. shafts	2	
No. segments	20	

The mass flow rate of water entering the Ragged Chutes downcomer selected is based upon the flow rate reported in Langborne (1979) and the mass flow rate of air entering the Ragged Chutes downcomer was determined based upon an average mass flow rate ratio of water to air of 1631 in HACs from historical data (Table 1.1)

Applying the inputs shown in Table 5.2, the model simulated the air absorption of each gas mixture in Table 5.1 including and excluding psychrometry logic. The mole fraction of oxygen in the gas phase at the downcomer outlet for each mixture is shown in Table 5.3.

Table 5.3 Results of gas mixture scenario testing

Mixture	Outlet oxygen dry mole fraction, χ' [mol / mol]	
	Psychrometry excluded	Psychrometry included
I	0.1781	0.1774
II	0.1783	0.1778
III	0.1779	0.1774
IV	0.0468	0.0466

The results shown in Table 5.3 indicate that the mole fraction of oxygen at the outlet of the downcomer is more dependent upon the mole fraction at the inlet than which gases are included in the mixture. This is expected as the air absorption formulation assumes that the limiting resistance is in the liquid phase and because the concentration of each gas is diffuse in the liquid (x_j in the order of $1x10^{-5}$) the diffusion of a given gas component is independent of the other gases dissolved in the liquid.

When simulating the absorption of atmospheric air at these mass flow rates of air and water, the dry mole fraction of oxygen at the outlet has good agreement with the measured value from Ragged Chutes reported by E.B.W. (1910) and McNair and Koenig (1911). The mole fraction of oxygen predicted by Chen and Rice (1983) simulating the same installation was 0.174 but it is not explicitly stated which mass flow rates were used as the sources they cite for the operating conditions of the HAC do not state what flow rate of water was typical for the installation. Psychrometry did not affect the results significantly because at this temperature the inlet mole fraction of water in the gas mixture is approximately 2% by mole at saturation conditions (100% relative humidity) and does not reduce the mole fraction of oxygen at the inlet significantly but this is likely to change at higher temperatures.

At 25°C and atmospheric pressure, the Henry's constants of nitrogen and carbon dioxide are 6.4×10^{-6} m^3/mol Pa and 3.4×10^{-4} m^3/mol Pa respectively (see Table 2.6). This demonstrates that carbon dioxide is more soluble in water than nitrogen by two orders of magnitude and the results of the downcomer model should reflect this. The concentration profiles of each gas in both phases were collected from the simulation results of the Ragged Chutes downcomer using mixture III to examine the rates at which each gas are dissolving into the water predicted by the model and is shown in Figure 5.1.

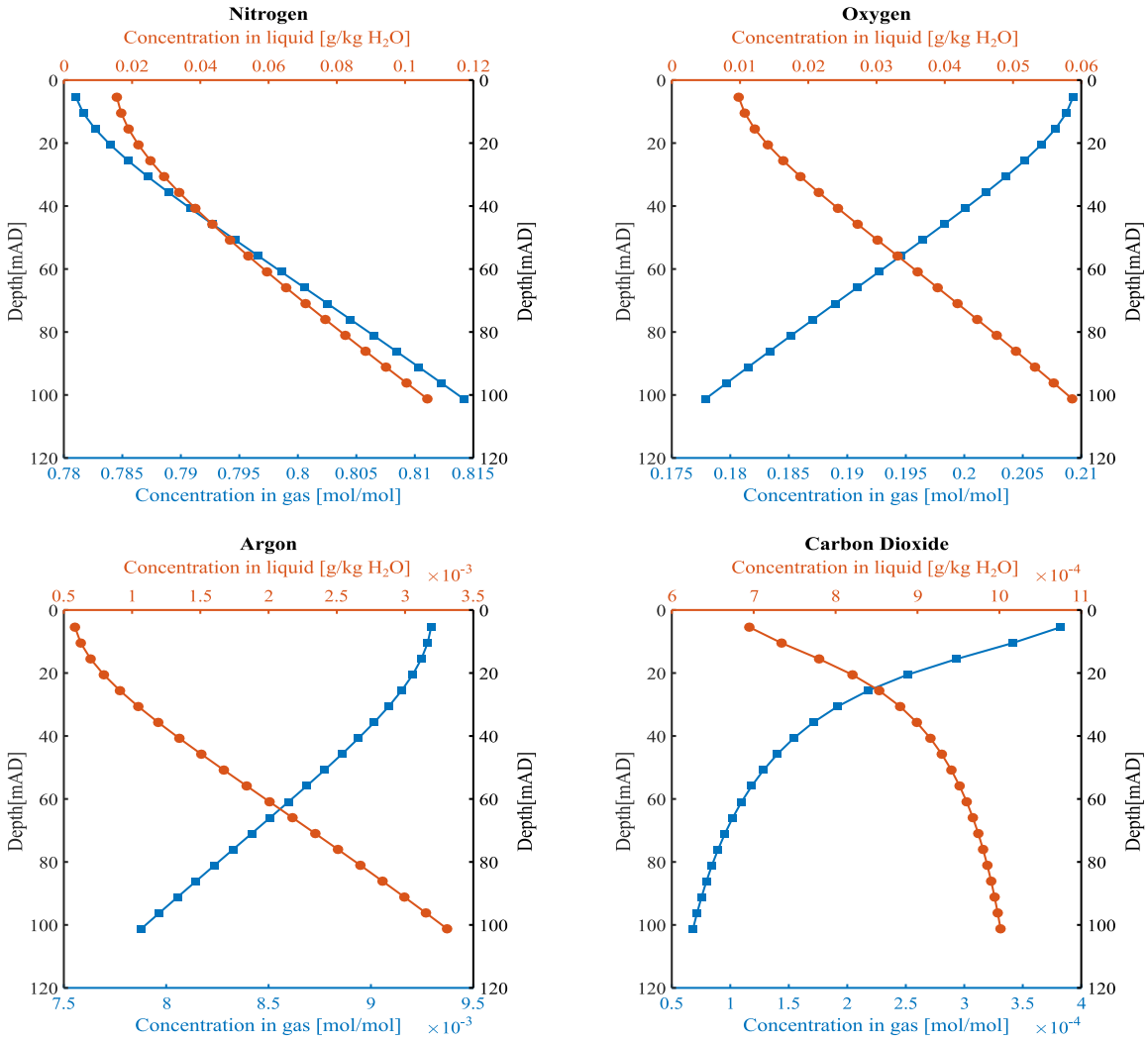


Figure 5.1 Concentration profiles of Mixture III in the Ragged Chutes downcomer

The concentration profiles show that the gases are being absorbed at different rates as expected from their Henry's solubility constants. The mole fraction of nitrogen increases as the flow descends the downcomer because it is the least soluble in water of the gas mixture components. The solubilities of oxygen, argon, and carbon dioxide being higher, these dissolve first, reducing the total moles of any/all species remaining in the gas phase. All the gases dissolve but the concentration of the remaining nitrogen increases because the other species dissolve faster having a higher driving force due to their solubilities.

The mass transfer coefficients, as evaluated by the Higbie (1935) relationship (2.31), for each of the species differ from each other by only their mass diffusivity in water since the exposure time is determined through the local gas slip velocity and bubble diameter. From Table 2.4, it can be seen that the mass diffusivities of nitrogen, oxygen, argon and carbon dioxide are similar in magnitude to each other and, consequently, so will the mass transfer coefficients.

5.2 Evaluating mesh independence

Since the downcomer model is a discrete approximation of a continuous process, it is necessary to evaluate the number of segments required to attain a mesh independent solution. Taking the model inputs from Table 5.2, approximating atmospheric air with gas mixture II in Table 5.1 and excluding psychrometry, the number of segments was flexed to determine at which point the model converges upon a mesh independent solution.

Table 5.4 Model convergence behaviour to mesh independence

No. segments	Outlet oxygen dry mole fraction, χ' [mol / mol]
20	0.1783
40	0.1781
60	0.1780
80	0.1780
100	0.1780
200	0.1779
400	0.1779
2000	0.1779

From Table 5.4 it can be seen that the model reaches mesh independence at 200 segments but the result at 20 segments has less than 1% relative error to the converged value. This suggests that 20 segments can be used with the downcomer model to reduce the required computational effort without sacrificing accuracy.

5.3 Mass flow rate ratio

The mass flow rate ratio of water to gas influences the air absorption in the downcomer shaft by affecting the bulk liquid concentration. If the mass flow ratio increases with a given fixed mass flow of gas, the increase in water mass flow rate will reduce the bulk liquid concentration of gases and will increase in the potential for air absorption. Another way of expressing this behaviour is that if the chemistry of the compressed gas is analysed, its composition depends on the mass flow rates of water and gas species at inlet.

Millar (2014) presented how the oxygen mole fraction in the compressed air delivered by the Victoria Mine HAC and the Ragged Chutes HAC was sensitive to the mass flow rate ratio and intake water temperature assuming that the compressed air has reached an equilibrium condition.

To ensure the model developed herein exhibits the same behaviour, the Ragged Chutes HAC downcomer was simulated using the inputs in Table 5.2 but using a mass flow rate of air of 22.7 kg/s, mass flow rate ratios of 1150, 1200, 1250, 1300, 1350 and 1400 to determine the water mass flow rate and excluding psychrometry.

Table 5.5 Mass flow ratio test results of Ragged Chutes HAC with gas mass flow rate of 22.7 kg/s and inlet temperature of 294.15 K

Mass flow rate ratio	$\chi'_{O2,out}$ [mol / mol]	Millar (2014) $\chi'_{O2,out}$ [mol / mol]	$C_{O2,B,out}$ [mol / m^3]	$A_{i,total}$ [m^2]
1150	0.1880	0.1810	1.8335	62.13
1200	0.1867	0.1800	1.8501	67.66
1250	0.1855	0.1790	1.8575	73.28
1300	0.1844	0.1780	1.8575	78.98
1350	0.1833	0.1770	1.8516	84.75
1400	0.1824	0.1760	1.8408	90.57

Millar (2014) re-evaluated the mass flow rate of water entering the Ragged Chutes HAC downcomer based upon the mass flow rate which would bring the mole fraction of oxygen to match the measured value of 0.177 (E.B.W., 1910; McNair and Koenig, 1911) at 294.15 K assuming that the gas was at equilibrium conditions. Millar (2014) determined that a mass flow rate of water of 30,590 kg/s would be required to achieve the measured value reported by E.B.W. (1910) and McNair and Koenig (1911).

For comparison with the equilibrium solubility model of Millar (2014), the Ragged Chutes HAC downcomer is simulated using a mass flow rate of water of 30,590 kg/s and using the same mass flow rate ratios in the previous test to determine the mass flow rate of gas with the model outlined herein excluding psychrometry.

Table 5.6 Mass flow ratio test results of Ragged Chutes HAC with liquid mass flow rate of 30,590 kg/s and inlet temperature of 294.15 K

Mass flow rate ratio	$\chi'_{O2,out}$ [mol / mol]	$C_{O2,B,out}$ [mol / m^3]	$A_{i,total}$ [m^2]
1150	0.1874	1.8551	73.05
1200	0.1864	1.8562	75.90
1250	0.1854	1.8559	78.77
1300	0.1843	1.8544	81.68
1350	0.1833	1.8518	84.61
1400	0.1824	1.8483	87.57

The same trend of increasing the mass flow rate ratio with a fixed mass flow rate of gas is exhibited by the model as shown in Table 5.5 but the mole fractions of oxygen are consistently higher when compared to the results from Millar (2014). When the mass flow rate ratio increases with a fixed liquid mass flow rate, mass flow rate of the gas is reduced which reduces the gas superficial velocity in the downcomer. While the absolute value differs, when either the gas mass flow rate or liquid mass flow rate are fixed, increasing the mass flow ratio tends to increase the total interfacial area in the downcomer shaft which is consistent the visualisations of downward bubbly flow from Bhagwat and Ghajar (2012) as shown in figures 2.7 and 2.8.

To determine the influence of the solubility kinetics on the results the Ragged Chutes HAC downcomer was simulated using a mass flow rate of air of 22.7 kg/s, a mass flow rate of water of 30,590 kg/s and the mass diffusivity of the gases was artificially increased by factors ranging from 0 to 2000 to determine how much the diffusivity would have to increase to for the model to predict equilibrium conditions at the bottom of the downcomer shaft. The results are tabulated in Table 5.7.

Table 5.7 Results of mass diffusivity sensitivity test

Diffusivity Factor	P_{out} [Pa]	$\dot{m}_{g,out}$ [kg/s]	$\chi'_{O2,out}$ [mol/mol]	$C_{O2,i,out}$ [mol/m^3]	$C_{O2,B,out}$ [mol/m^3]	$C_{O2,B,out}$ [% sat.]
0	896,753	22.70	0.2095	2.612	0.296	11.32%
1	907,947	18.07	0.1820	2.297	1.933	84.16%
5	910,734	17.40	0.1799	2.278	2.109	92.58%
10	911,574	17.23	0.1794	2.274	2.153	94.69%
15	911,974	17.15	0.1792	2.272	2.173	95.64%
20	912,221	17.10	0.1791	2.271	2.185	96.21%
30	912,524	17.04	0.1790	2.270	2.200	96.89%
40	912,709	17.01	0.1789	2.270	2.208	97.30%
50	912,838	16.99	0.1788	2.269	2.214	97.58%
100	913,164	16.93	0.1787	2.268	2.229	98.28%
150	913,312	16.90	0.1786	2.268	2.236	98.60%
200	913,401	16.88	0.1786	2.268	2.240	98.78%
250	913,469	16.87	0.1786	2.267	2.243	98.91%
300	913,515	16.86	0.1785	2.267	2.245	99.00%
500	913,624	16.84	0.1785	2.267	2.249	99.23%
2000	913,820	16.81	0.1784	2.266	2.258	99.64%

Here the bulk concentration of oxygen, $C_{O2,B,out}$, in terms of percent saturation is simply the ratio of the concentration of oxygen in the bulk fluid to the concentration of oxygen at the gas liquid interface (equilibrium concentration).

When the model approaches equilibrium, it converges on a different concentration than what is reported by Millar (2014). The values adopted for the Henry's constants by Millar were different from those adopted in this work (see Table 5.8). This discrepancy was due different sources used for the selection of the values Henry's constants. Millar (2014) selected values from Sander (1999) while the values were selected from Sander (2015) for the purposes of this work.

Table 5.8 Comparison of parameters selected in Millar (2014) and this work for use in the van 't Hoff equation.

Species	Millar (2014)		This work	
	$H_{j,0}^{cp}$ [mol/m^3 Pa]	$-\Delta\check{h}_{sol}/_{\mathcal{R}}$ [K]	$H_{j,0}^{cp}$ [mol/m^3 Pa]	$-\Delta\check{h}_{sol}/_{\mathcal{R}}$ [K]
Nitrogen	6.02E-06	1300	6.40E-06	1300
Oxygen	1.28E-05	1500	1.30E-05	1500
Argon	1.38E-05	1100	1.40E-05	1500
Carbon dioxide	3.45E-04	2200	3.40E-04	2400

When the values selected by Millar (2014) are adopted in the model the equilibrium convergence on the same equilibrium as predicted by Millar (2014).

Table 5.9 Diffusivity sensitivity test results adopting the Henry's constants values from Millar (2014)

Diffusivity Factor	P_{out} [Pa]	$\dot{m}_{g,out}$ [kg/s]	$\chi'_{O2,out}$ [mol/mol]	$C_{O2,i,out}$ [mol/m^3]	$C_{O2,B,out}$ [mol/m^3]	$C_{O2,B,out}$ [% sat.]
0	896,753	22.70	0.2095	2.578	0.292	11.32%
1	907,452	18.27	0.1813	2.257	1.902	84.27%
5	910,103	17.64	0.1791	2.237	2.072	92.65%
10	910,901	17.47	0.1786	2.233	2.115	94.75%
15	911,280	17.40	0.1784	2.231	2.135	95.69%
20	911,515	17.35	0.1783	2.230	2.146	96.25%
30	911,803	17.30	0.1781	2.229	2.160	96.93%
40	911,978	17.27	0.1781	2.228	2.168	97.33%
50	912,100	17.24	0.1780	2.227	2.174	97.61%
100	912,410	17.19	0.1778	2.226	2.189	98.30%
150	912,550	17.16	0.1778	2.226	2.195	98.61%
200	912,635	17.15	0.1777	2.226	2.199	98.80%
250	912,699	17.13	0.1777	2.225	2.201	98.92%
300	917,626	17.10	0.1775	2.235	2.213	99.01%
500	912,846	17.11	0.1776	2.225	2.208	99.24%
2000	913,033	17.08	0.1775	2.224	2.216	99.64%

The potential causes of the discrepancies between the model and the Millar (2014) equilibrium model are the outlet pressure and the solubility kinetics. Since the downcomer process model, when it is decoupled from the HAC system, does not have a pressure boundary condition, the pressure at the outlet of the downcomer may not match the delivery pressure of Ragged Chutes but upon inspection of tables 5.7 and 5.9 it can be seen that the model predicts an outlet pressure close to the reported delivery pressure of Ragged Chutes of 908,000 Pa absolute. In order to

reach equilibrium solubility conditions at the downcomer outlet, the mass diffusivities needed to be increased by a factor of 2,000. This suggests that the solubility kinetics are a significant factor and that the equilibrium conditions were reached in the 300 m long separation gallery of Ragged Chutes.

5.4 Depth of downcomer outlet

Further testing was conducted with the model to simulate a downcomer of the type proposed for tidal power recovery (Millar et al., 2016). Using the inputs shown in Table 5.10, the downcomer was simulated with lengths of 1, 10, 20, 50, 100 and 200 m and mass flow rate ratios of water to air of 500, 1000, 2000, 7000 and 20,000.

Table 5.10 Downcomer model inputs for gas yield tests based upon inputs from Millar et al. (2016)

Input parameter	Value	Units
Inlet water mass flow rate	900	kg/s
Inlet pressure	101,032	Pa
Inlet temperature	283.15	K
Shaft inside diameter	0.575	m
Flow direction	-90	°
Absolute roughness	$1x10^{-3}$	m
Properties used to evaluate wall shear stress	Liquid phase only	
No. shafts	1	
No. segments	100	

At each combination of downcomer length and mass flow rate ratio, the gas yield of each gas component was evaluated for mixtures III and IV in Table 5.1. The yield of an individual gas species was defined as the ratio of the molar flow rate of the species in the gas phase at the outlet

in comparison to that at the inlet and the results are tabulated in Appendix E. Because nitrogen is less soluble than carbon dioxide, gas solubility increases with pressure, and Millar (2014) has established that increasing the mass flow rate ratio increases the amount of gas dissolved in the downcomer of a HAC, the yield of nitrogen should be higher than carbon dioxide in each scenario and the yield of both nitrogen and carbon dioxide should decrease with increasing downcomer length and mass flow rate ratio. The resulting gas yield contours at different downcomer lengths and mass rate ratios of nitrogen and oxygen for mixture III and mixture IV are shown in figures 5.2 and 5.3 respectively.

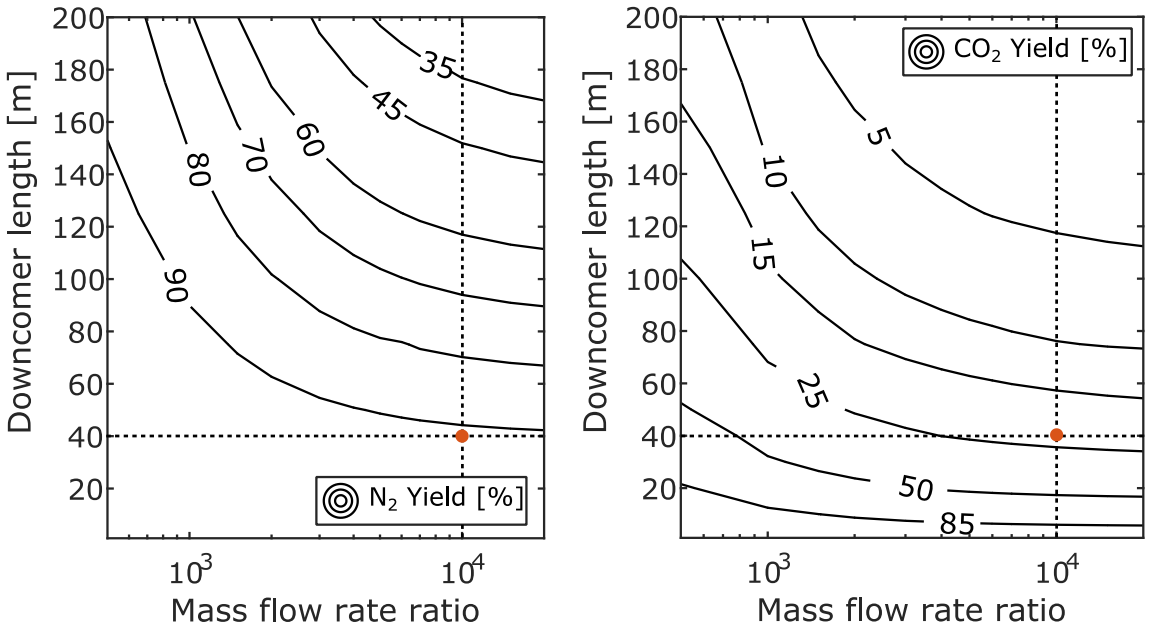


Figure 5.2 Computed gas yield contours for absorption of atmospheric air at different downcomer lengths and mass flow rate ratios

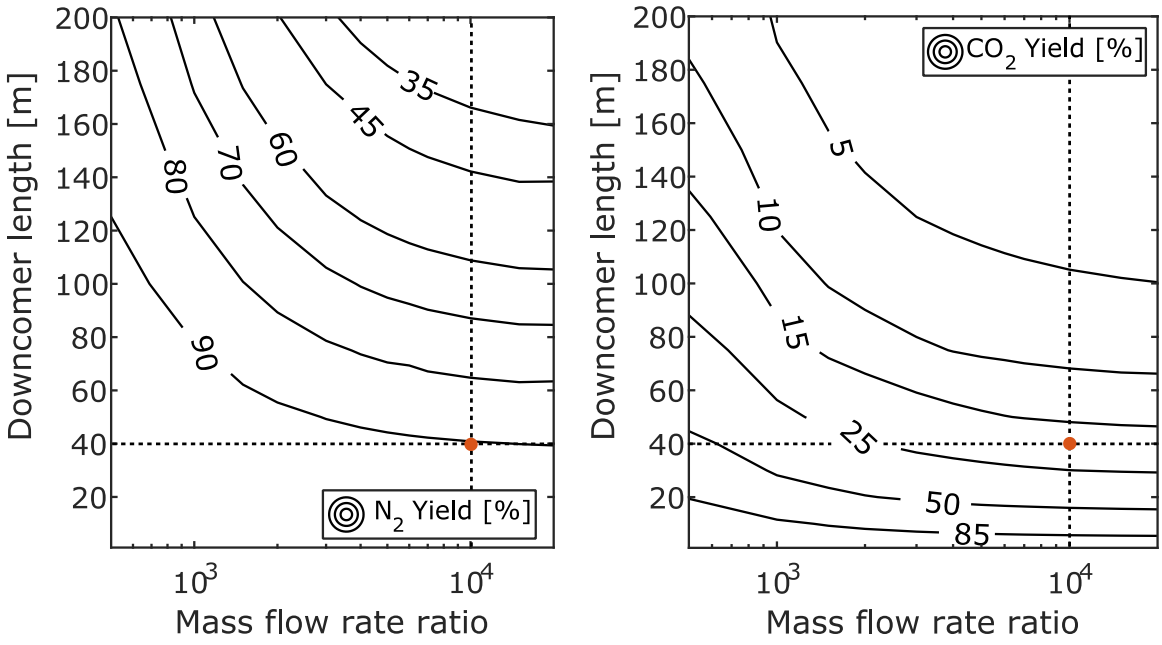


Figure 5.3 Computed gas yield contours for absorption of flue gas at different downcomer lengths and mass flow rate ratios

For the absorption of air in a downcomer that is 40 m long and has a mass flow rate ratio of 10,000, the model predicts a nitrogen yield of just above 90% and a carbon dioxide yield of 25%. While for the absorption of flue gas in a downcomer with the same length and mass flow rate ratio, the model predicts a nitrogen yield of 90% and a carbon dioxide yield of under 25%. The full results of the gas yield scenario testing are tabulated in Appendix E.

The gas yield contours shown in figures 5.2 and 5.3 indicate that the gas yield is reduced as the mass flow rate ratio and the downcomer length increase and that carbon dioxide is absorbed more readily than nitrogen as expected.

When the downcomer model is decoupled from the HAC system model, increasing the depth of the downcomer outlet will increase the pressure of the fluids. Higher pressures of the fluid will

promote more air to be absorbed by the water due to the increased interfacial concentration. Using the downcomer dimensions in Table 5.10, a mass flow rate ratio of 2000 and discretizing the downcomer in to 20 segments, the model was used to simulate vertical downcomers with depths of 1, 10, 20, 50, 100, and 200 m.

Table 5.11 Downcomer outlet depth scenario results

Depth [m]	Outlet Absolute Pressure [Pa]	$\chi'_{O2,out}$ [mol / mol]	$C_{O2,B,out}$ [mol / m^3]
1	107,861	0.2095	0.3595
10	173,321	0.2087	0.3919
20	251,873	0.2064	0.4778
50	506,088	0.1954	0.8931
100	955,059	0.1728	1.6809
200	1,882,650	0.1316	2.8859

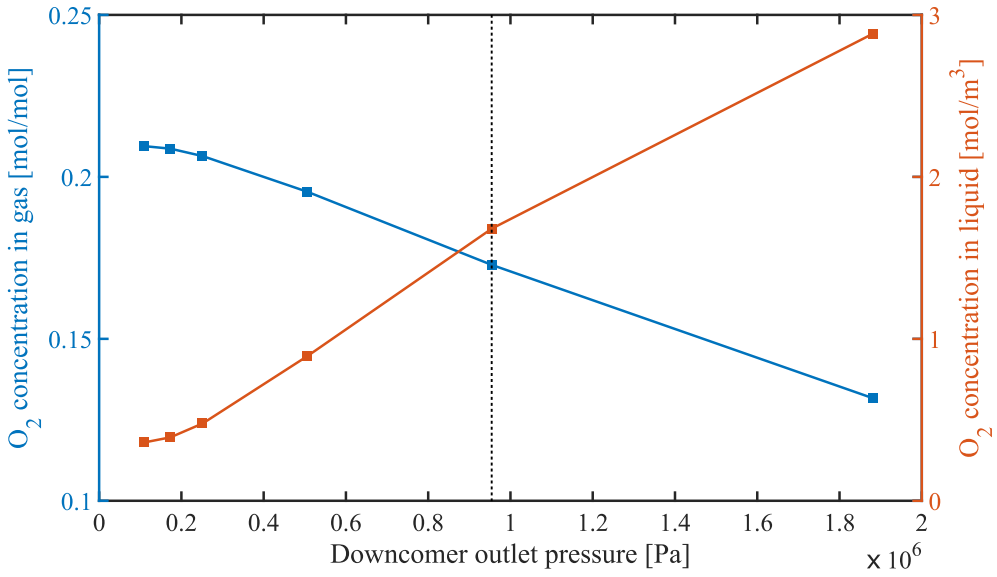


Figure 5.4 Downcomer outlet pressure versus oxygen concentration in the gas and liquid phases. The vertical dashed line corresponds to the Ragged Chutes downcomer depth.

As expected, the model shows that an increase in the depth leads to a reduction of the mole fraction of oxygen at the outlet of the downcomer and a corresponding increase concentration in the liquid. Note that the downcomer depth of 100 m corresponds to downcomer of the Ragged Chutes HAC installation.

5.5 Intake liquid temperature

Once the air and water phases are completely mixed, the temperature of the air quickly becomes the temperature of the water (Chen and Rice, 1983) due the larger thermal mass of the water. Consequently the air absorption in the downcomer is sensitive to the intake liquid temperature rather than the intake temperature of the air (Millar, 2014). The compression in the downcomer is also nearly isothermal so that there is only a slight increase in temperature from the inlet to the outlet of the downcomer (Pavese, 2015). When the water temperature increases it affects the air absorption in the downcomer shaft in three ways: i) reducing the Henry's solubility constant ii) increases the absolute humidity of the air and iii) increases the mass diffusivity of the gas solutes in liquid.

Based upon the van 't Hoff equation (2.47), as the water temperature increases the Henry's solubility constant and hence the equilibrium solubility of gases in water decrease. This serves to reduce the driving force for air absorption in water by reducing the interfacial concentration. When the air absorption is occurring in the downcomer of the HAC where the gas is constantly at 100% relative humidity, the absolute humidity of the gas increases non-linearly with the temperature of the water as shown in Figure 5.5

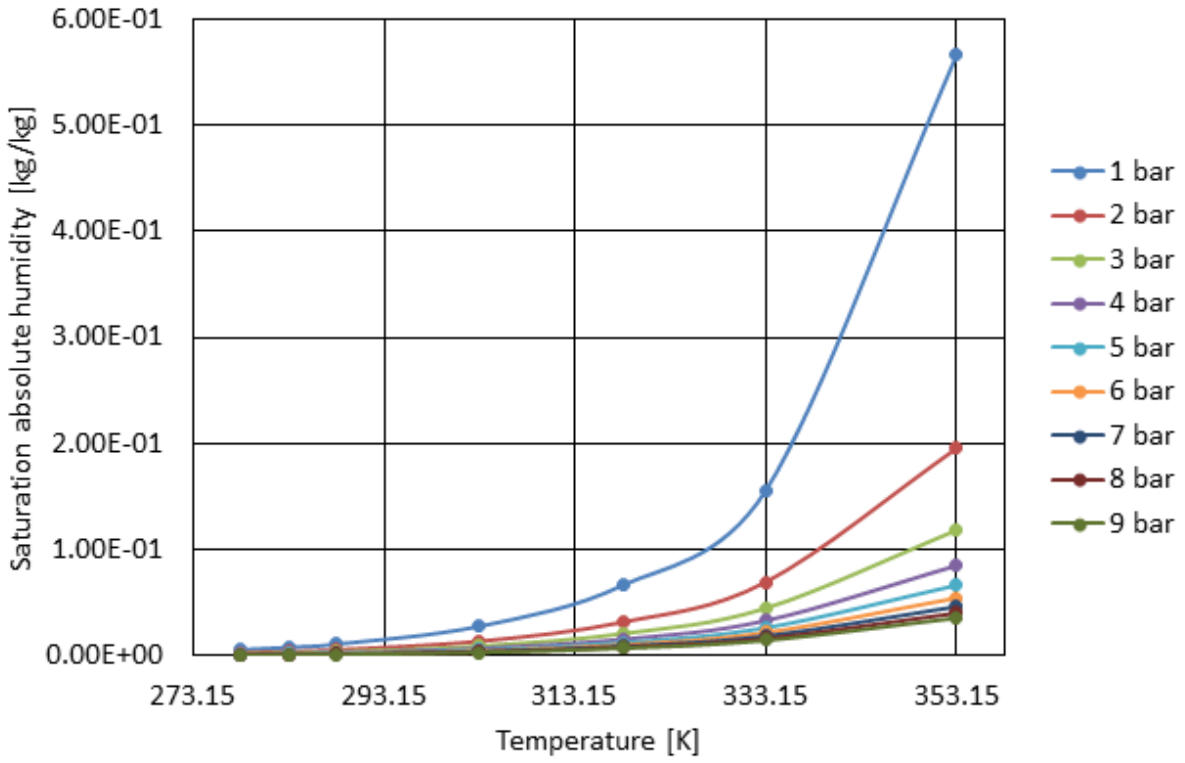


Figure 5.5 Saturation absolute humidity with increasing temperature and pressure

The increase in absolute humidity reduces the partial pressure of the dry atmospheric gases because more of the mixture is water vapour. At higher temperatures it is easier for gases to move through water through molecular diffusion and this is accounted for by adjusting the mass diffusivity of gases with the Stokes-Einstein equation (2.34).

The downcomer model was used to simulate a downcomer with the inputs shown in Table 5.10, a mass flow rate ratio of 2,000 and a length of 100 m and inlet temperatures of 5, 10, 15, 30, 45, 60 and 80 °C. Each scenario was simulated including and excluding the psychrometry logic in the model to examine both effects of increasing the water temperature.

Table 5.12 Dry scenario test results (excludes psychrometry)

Inlet Temperature [K]	Outlet Temperature[K]	$\chi'_{O2,out}$ [mol / mol]	$C_{O2,B,out}$ [mol / m^3]
278.15	278.189	0.1717	1.7294
283.15	283.194	0.1728	1.6809
288.15	288.197	0.1741	1.6276
303.15	303.207	0.1788	1.4461
318.15	318.215	0.1836	1.2604
333.15	333.223	0.1879	1.0884
353.15	353.233	0.1925	0.8914

Table 5.13 Humid scenario test results (includes psychrometry)

Inlet Temperature [K]	Outlet Temperature [K]	$\chi'_{O2,out}$ [mol/mol]	$C_{O2,B,out}$ [mol/m^3]	γ'_{out} [kg/kg]
278.15	278.191	0.1715	1.7244	0.0006
283.15	283.195	0.1725	1.6749	0.0008
288.15	288.199	0.1738	1.6166	0.0011
303.15	303.214	0.1780	1.4300	0.0028
318.15	318.230	0.1820	1.2322	0.0065
333.15	333.255	0.1846	1.0417	0.0137
353.15	353.318	0.1832	0.8013	0.0335

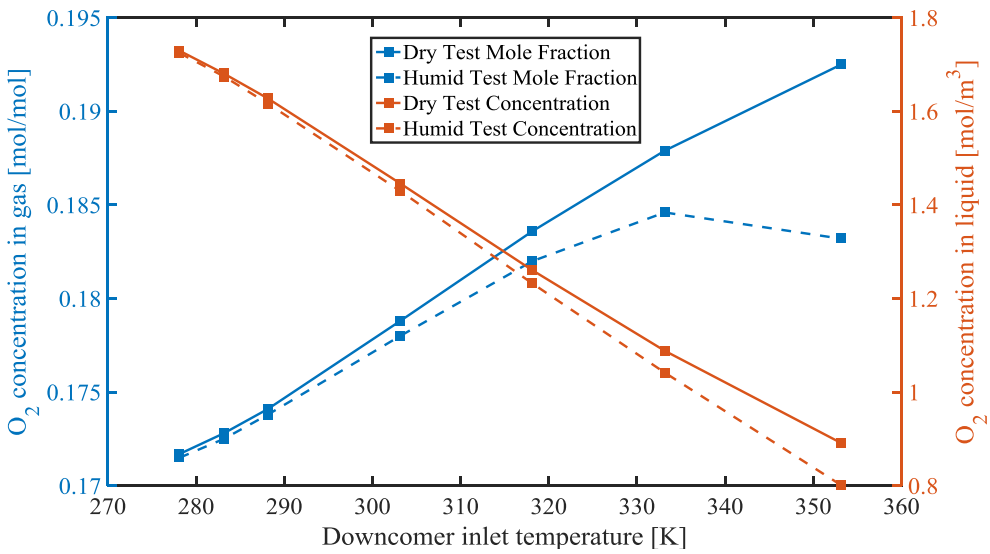


Figure 5.6 Downcomer inlet temperature versus oxygen concentration at the outlet in both phases

From tables 5.1 and 5.1 and Figure 5.6 it can be seen that the model shows a reduction in the concentration of oxygen in the liquid phase at increased temperatures and a corresponding increase in the gas phase. When psychrometry is included even less oxygen is dissolved in the liquid. Figure 5.5 shows that the saturation moisture content of air is strongly non-linear above 320 K, particularly when the air pressure is low. It is this behaviour that influences the oxygen concentration in the gas phase at temperatures above 320 K, exhibited as a significant divergence between the curve for dry gas and humid gas.

5.6 Co-solute concentration

As previously stated in section 4.6, presence of electrolytes in solution of the liquid phase is expected to affect the air absorption in the downcomer by reducing the Henry's solubility constants and mass diffusivities of the gases in the liquid phase. Since the presence of electrolytes also affect the fluid properties of the liquid phase the co-solutes that used in the

scenarios to test the downcomer formulation were based upon those available with the thermodynamic software CoolProp (Bell et al., 2014) as shown in Table 4.2.

To evaluate the sensitivity of the mole fraction of oxygen in the gas phase to co-solute composition and concentration, the downcomer shown in Table 5.10 was simulated with a length of 100 m, a mass flow rate ratio of 2,000, and psychrometry logic turned off while co-solutes of sodium chloride, magnesium chloride and potassium carbonate were added with mass fractions from 0.02 to 0.16 at intervals of 0.02. Because the mass diffusivity, Henry's constant and Sechenov constant are all dependent upon temperature, inlet temperatures of 294.15 K and 308.15 K were used to show the influence of temperature on the effectiveness of the co-solute. The full results of the co-solute tests are tabulated in Appendix F and a selection are shown in the following figure.

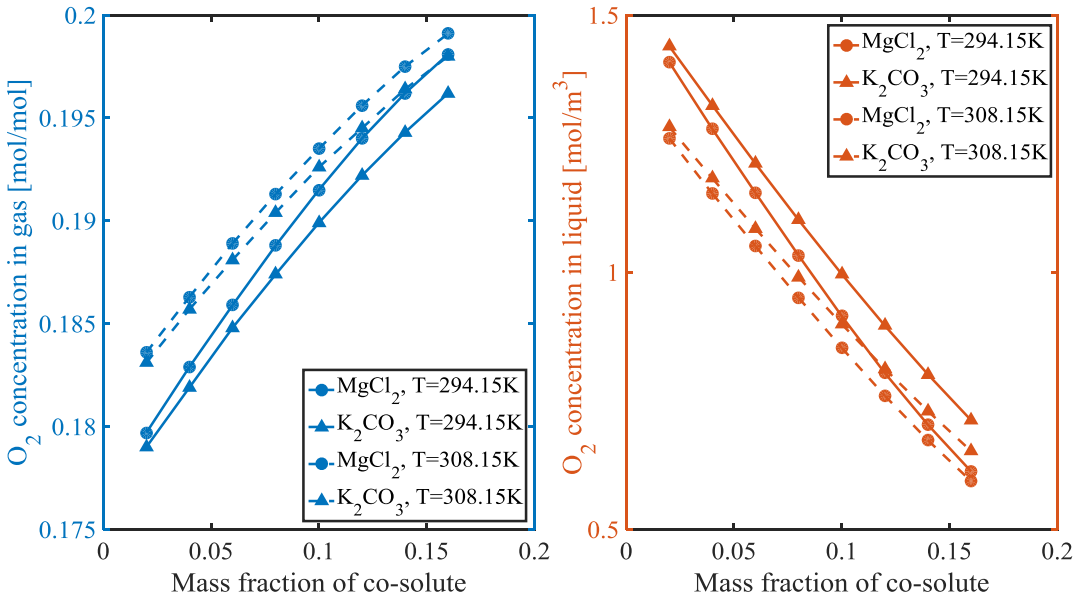


Figure 5.7 Co-solute test results for varying concentrations of magnesium chloride and potassium carbonate at 294.15 and 308.15 K

From Figure 5.7 it can be seen that magnesium chloride was the more effective co-solute providing the required behaviour of at salting-out oxygen than potassium carbonate. At 308.15 K a marginal increase in co-solute concentration produces a lower marginal reduction in oxygen concentration.

The model had difficulties modelling co-solutes with a mass fraction greater than 0.18 because when the model would flex the molar flow rate of the species in the gas phase by a maximum of 2% with Newton-Raphson algorithm, this 2% would be more than the molar flow rate of the species in the liquid phase. This would trigger an error in the code as the model would be numerically creating mass. A potential solution for this issue would be to have the molar flow rate of the species in the liquid phase be part of the state variables which are flexed by the Newton-Raphson algorithm instead of the molar flow rates of the species in the gas phase. This correction would conceptually align with the two resistance theory of gas absorption since the air absorption in the HAC downcomer is limited by the liquid phase.

Chapter 6

6. Discussion

The downcomer model was tested with multiple scenarios which change the air absorption parameters in the HAC downcomer. In every case the downcomer model behaved consistently with expectation, and where variances emerged, these became understood when the details of the various interactions were clarified.

The model permits the prediction of the reduction of compressed air yield due to solubility. The model can predict the improvement of compressed air yield from the interventions of increasing the liquid temperature and introducing a co-solute.

6.1 The influence of psychrometry

Numerical instability was encountered with the downcomer model when psychrometry was included. When the downcomer model ran successfully with psychrometry, more computational effort would be required than if psychrometry was excluded. Hence the preference is to model the downcomer process neglecting the psychrometric equations if possible. At temperatures ranging from 278.15 K to 318.15 K, the results in tables 5.12 and 5.13 show that the solution predicted by the model when psychrometry is included is less than 1% different than when psychrometry is excluded. This suggests that in this temperature range, psychrometry can safely be neglected.

It appears that in the context of air absorption in the downcomer shaft as predicted by the model, the influence of psychrometry does not become preeminent until examining liquid temperatures

above 318.15 K (45 °C) (Figure 5.6). When comparing the dry to humid cases, the model did show a reduction in the liquid phase concentration of oxygen as expected but it was not met with a corresponding increase in the gas phase due to gas borne humidity. In every case the outlet concentration of oxygen in the gas phase was lower in the humid case than the dry case.

The cause of this behaviour in the model stems from how the gas phase composition is initialised at the downcomer inlet. The mass flow of the gas phase in the humid case includes the amount of water vapour in the air at saturation conditions at the inlet temperature. At increasing temperatures more of the mass flow is taken up by water vapour. While the mass of each species is conserved in the downcomer, the total amount of oxygen in the downcomer is reduced with increasing temperature when the composition is initialised in this way. This leads to the reduction of mole fraction of oxygen and makes it difficult to compare the scenarios.

In a further refined model the composition of air when including psychrometry should be initialised based upon given atmospheric conditions of air temperature and relative humidity. Then the water that is evaporated to bring the air to saturation conditions at the downcomer inlet should be treated as a mass transfer during the mixing process. This would provide a fixed amount of dry air as a baseline to provide better comparisons.

6.2 Modelling bubble drag

As previously stated, there are many equations which approximate the established drag curve for particles travelling in a fluid. The approach to model drag adopted in the model developed in this work was that of Rowe and Henwood (1961). This drag model was selected because the aim of

this work was to replicate the approach of Chen and Rice (1983) so that the modelling results could be compared.

However, there are refined drag models which follow the drag curve more accurately (Cheng, 2009) and other approaches to modify the drag coefficient for bubble swarms (Roghair et al., 2011) that were not available to Chen and Rice when they developed their model. If the model developed in this work requires accuracy to reproduce experimental results in the future, the drag coefficient model is a potential source of inaccuracy which could be upgraded with a more accurate drag model.

6.3 Evaluating fluid properties

A sample of a profile of the downcomer model generated from MATLAB is shown in Appendix B. The profile shows the time spent in each function of the code and upon inspection of the table it can be seen that 63.6% of the total time simulating the downcomer flow is evaluating fluid properties with CoolProp. When the psychrometry logic is included in the simulation, the saturation vapour pressure of water needs to be evaluated on top of the base case. Additionally, if the Roghair et al. (2011) modification is implemented to modify the drag coefficient, it would require the surface tension of water to be evaluated which would further increase the amount of time spent evaluating fluid properties.

Since the simulation time is shown in Appendix B is only for evaluating one flow through of the downcomer and multiple evaluations of the downcomer are required to evaluate the entire HAC system, any improvement to the computational speed of modelling the downcomer should start by improving the speed of evaluating fluid properties. The ways in which the computational

effort required to evaluate the fluid properties can be reduced include: i) evaluation of the fluid properties by interpolating from precomputed property tables and ii) evaluation of the fluid properties through CoolProp's low-level interface.

Interpolating the fluid properties from precomputed property tables could reduce the computational effort of simulating the downcomer by front loading computation effort to evaluate the fluid properties in a range of pressures and temperatures before the downcomer flow is evaluated. Then during the downcomer simulation the code interfaces with precomputed property tables instead of the CoolProp software which should require less time. This would require property tables for the density, viscosity and internal energy of nitrogen, oxygen, argon, carbon dioxide and water. To evaluate the properties of the whole gas phase a mass or mole fraction weighted average of the properties of nitrogen, oxygen, argon and carbon dioxide interpolated from the tables.

The current state of the code evaluates the fluid properties through CoolProp's high-level interface but there is a low-level interface which can be used to evaluate multiple fluid properties but bypassing some of the overhead associated with the high-level interface. Provided the low-level interface can be implemented in such a way that can take advantage of the reduced overhead per call it would reduce the computational effort required to evaluate the fluid properties.

6.4 Completing the loop

The results presented in Chapter 5 were simulations of a downcomer that is decoupled from the entire HAC system. This allowed for more freedom in the selection of the mass flow rate of gas

in the downcomer but when the downcomer forms part of the HAC system, this is not the case. In the operation of a HAC only the mass flow rate of liquid can be controlled and the mass flow rate of gas is that which the flow losses (including gas compression) through the system bring the pressure at the tailrace elevation back to atmospheric pressure. In order for the current code to solve the HAC system as a whole three things need to be added: i) a more refined model of the air-water mixing, ii) mass transfers in the air-water separator and iii) a class to numerically evaluate the mass flow rate of gas inducted by the HAC.

The purpose of the air-water mixing domain in the current formulation is to simply evaluate the state of the fluids at the downcomer inlet but the flow losses and mixing irreversibilities through the domain are not evaluated. A more refined air-water mixing class would be one which evaluates the flow loss and the mass transfer from the evaporation of water so that the state of the fluids at the inlet of the downcomer can be determined. The irreversibility from mixing is also not considered in the current methodology but this is expected to be one of the deliverables of the research done by Hutchison (2016). Additionally, a more mechanistic model of determining the point at which the fluid pressure recovers to atmospheric in the downcomer instead of taking it as a design parameter would also be considered an asset.

The current methodology adopted for the separator neglects the interphase mass transfers that occur in the separator and assumes that the composition of the compressed air delivered by the HAC is the composition of the gas at the outlet of the downcomer. To fully couple the solubility kinetics, psychrometrics and the hydrodynamics the mass transfers in the air-water separator would have to be added to the modelling of the separator especially because the air-water mixture has a high residence time in the separation chamber in both the Ragged Chutes HAC and

the Peterborough HAC designs. A more mechanistic model of the pressure drop across the separator would be considered an asset.

Finally, to model the HAC system as a whole, a class is required which orchestrates the numerical determination of the mass flow rate of gas inducted by the HAC and ensures that the pressure at the tailrace elevation returns to atmospheric pressure. Most HAC designs had a flare at the base of the downcomer to reduce the flow velocities entering the air-water separator. While the formulation in this work can handle divergent ducts, the HAC class would have to be able to handle more than one instance of the downcomer class. The computation time to solve the HAC system is also an important consideration since it is desired for the HAC model to be used for model predictive control.

Chapter 7

7. Conclusions and future work

7.1 Conclusions

The model reported herein couples the solubility kinetics and psychrometrics with the hydrodynamics in the downcomer shaft of a HAC and its formulation has been fully detailed. The model permits the prediction of the reduction of compressed air yield due to solubility of gases in the downcomer shaft and the improvement of compressed air yield from increased water temperatures and the presence of a salt co-solute.

The model has been used in multiple trial evaluations to ensure that it returns behaviour consistent with that expected when adjustments to parameters which affect air absorption in the downcomer shaft are made. The model exhibits the expected behaviours of the components of air absorbing at different rates depending how soluble they are in water. In general, more gas is dissolved at higher outlet pressures of the downcomer and higher mass flow rate ratios; less gas is absorbed at higher liquid temperatures and co-solute concentrations.

The Ragged Chutes HAC downcomer was simulated at varying mass flow rates of water and air. Simulation results are presented which reproduce the mole fraction of oxygen of 0.177 (mol/mol) measured and reported by E.B.W. (1910) and McNair and Koenig (1911) at the outlet of the downcomer in two different HACs. The sensitivity of the mole fraction of oxygen with mass flow rate ratio in the Ragged Chutes downcomer is evaluated with the model developed in this work and consistently achieved a higher mole fraction of oxygen than the equilibrium model in

the work reported by Millar (2014). In order to achieve equilibrium conditions at the outlet of the downcomer, the mass diffusivities of nitrogen, oxygen, argon and carbon dioxide had to be increased from their values reported in the literature by a factor of 2000. This suggested that potential for gas to dissolve remained at the outlet of the downcomer and the equilibrium conditions were reached in the separation cavern.

7.2 Future work

Future work to be conducted from this work is to

1. Refine the air-water mixing formulation to include a mass transfer of water to reach saturation moisture content in the gas phase at the downcomer inlet
2. Refine the air-water separation formulation to include the solubility kinetics during the gas-liquid separation process
3. Complete the loop and produce a model which solves for the HAC system as a whole
4. Perform a measurement and verification program on a prototype and pilot plant HACs to establish the accuracy of the model.
5. Improve the efficiency of the code for the downcomer model such that it can be used for model predictive control.

Figure 7.1 5 m high Prototype HAC (left) and 30 m high pilot plant HAC (right)

References

Akita, K., 1981. Diffusivities of gases in aqueous electrolyte solutions. Ind. Eng. Chem. Fundam. 20, 89–94. doi:10.1021/i100001a017

Akita, K., Yoshida, F., 1974. Bubble Size, Interfacial Area, and Liquid-Phase Mass Transfer Coefficient in Bubble Columns. Ind. Eng. Chem. Process Des. Dev. 13, 84–91. doi:10.1021/i260049a016

Andrade, J., 2013. Solubility Calculations for Hydraulic Gas Compressors.

Arnold, K., Stewart, M., 2008. Surface production operations. Volume 1, Design of oil handling systems and facilities, 3rd ed. Elsevier.

Bell, I.H., Wronski, J., Quoilin, S., Lemort, V., 2014. Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp. Ind. Eng. Chem. Res. 53, 2498–2508. doi:10.1021/ie4033999

Benson, B.B., Jr., D.K., 1976. Empirical laws for dilute aqueous solutions of nonpolar gases. J. Chem. Phys. 64, 689. doi:10.1063/1.432215

Bhagwat, S.M., Ghajar, A.J., 2012. Similarities and differences in the flow patterns and void fraction in vertical upward and downward two phase flow. Exp. Therm. Fluid Sci. 39, 213–227. doi:10.1016/j.expthermflusci.2012.01.026

Bidini, G., Grimaldi, C.N., Postrioti, L., 1999. Performance analysis of a hydraulic air compressor. Proc. Inst. Mech. Eng. Part A J. Power Energy 213, 191–203.

doi:10.1243/0957650991537545

Brennen, C.E., 2005. Fundamentals of Multiphase Flow. Cambridge University Press.

Brinkman, R., Margaria, R., Roughton, F.J.W., 1933. The Kinetics of the Carbon Dioxide-Carbonic Acid Reaction. Philos. Trans. R. Soc. London 232, 65–97.

Brown, P.P., Lawler, D.F., 2003. Sphere Drag and Settling Velocity Revisited. J. Environ. Eng. 129, 222–231. doi:10.1061/(ASCE)0733-9372(2003)129:3(222)

Cai, J., Chen, T., Ye, Q., 1997. Void fraction in bubbly and slug flow in downward air-oil two phase flow in vertical tubes, in: International Symposium on Multiphase Flow, Beijing.

Cengel, Y.A., Cimbala, J.M., 2010. Fluid Mechanics - Fundamentals and Applications, 2nd ed. McGraw-Hill, New York.

Chen, L.-T., Rice, W., 1983. Properties of Air Leaving a Hydraulic Air Compressor (HAC). J. Energy Resour. Technol. 105, 54. doi:10.1115/1.3230876

Chen, L.-T., Rice, W., 1982. Some Psychometric Aspects of a Hydraulic Air Compressor (HAC). J. Energy Resour. Technol. 104, 274. doi:10.1115/1.3230415

Cheng, N.-S., 2009. Comparison of formulas for drag coefficient and settling velocity of spherical particles, Powder Technology. doi:10.1016/j.powtec.2008.07.006

Clift, R. (Roland), Grace, J.R., Weber, M.E., 1978. Bubbles, drops, and particles. Academic Press.

Diamond, L.W., Akinfiev, N.N., 2003. Solubility of CO2 in water from −1.5 to 100 °C and from

0.1 to 100 MPa: evaluation of literature data and thermodynamic modelling. Fluid Phase Equilib. 208, 265–290. doi:10.1016/S0378-3812(03)00041-4

Duan, Z., Sun, R., 2003. An improved model calculating CO2 solubility in pure water and aqueous NaCl solutions from 273 to 533 K and from 0 to 2000 bar. Chem. Geol. 193, 257–271. doi:10.1016/S0009-2541(02)00263-2

Dumaresq, C., 2009. Cobalt Mining Legacy [WWW Document]. URL http://www.cobaltmininglegacy.ca/power.php (accessed 11.3.16).

E.B.W., 1910. Hydraulic air compression. Mines Miner. 31, 129–131.

Eastop, T.D., McConkey, A., 1993. Applied Thermodynamics for Engineering Technologists, 5th ed. Pearson Education Ltd.

Geng, M., Duan, Z., 2010. Prediction of oxygen solubility in pure water and brines up to high temperatures and pressures. Geochim. Cosmochim. Acta 74, 5631–5640. doi:10.1016/j.gca.2010.06.034

Government of Quebec, 2013. Usine Dominion Textile - Répertoire du patrimoine culturel du Québec [WWW Document]. URL http://www.patrimoine-culturel.gouv.qc.ca/rpcq/detail.do?methode=consulter&id=105330&type=bien#.WMvqcW_yttR (accessed 3.17.17).

Hermann, C., Dewes, I., Schumpe, A., 1995. The estimation of gas solubilities in salt solutions, Chemical Engineering Science. Pergamon. doi:10.1016/0009-2509(95)00031-Y

Higbie, R., 1935. The rate of absorption of a pure gas into a still liquid during short periods of exposure. Trans. Am. Inst. Chem. Eng. 35, 365–389.

Himmelblau, D.M., 1964. Diffusion of Dissolved Gases in Liquids. Chem. Rev. 64, 527–550. doi:10.1021/cr60231a002

Hutchison, A., 2016. Air-water mixing and separation in hydraulic air compressors.

Jamnongwong, M., Loubiere, K., Dietrich, N., Hébrard, G., 2010. Experimental study of oxygen diffusion coefficients in clean water containing salt, glucose or surfactant: Consequences on the liquid-side mass transfer coefficients. Chem. Eng. J. 165, 758–768. doi:10.1016/j.cej.2010.09.040

Johansen, O., Rye, H., Melbye, A.G., Jensen, H. V., Serigstad, B., Knusten, T., 2000. Deep Sill JIP - Experimental Discharges of Gas and Oil at Helland Hansen - June 2000, Technical Report.

Jones, G., Dole, M., 1929. The viscosity of aqueous solutions of strong electrolytes with special reference to barium chloride. J. Am. Chem. Soc. 51, 2950–2964. doi:10.1021/ja01385a012

Kantarci, N., Borak, F., Ulgen, K.O., 2005. Bubble column reactors. Process Biochem. 40, 2263–2283. doi:10.1016/j.procbio.2004.10.004

Karamanev, D.G., Nikolov, L.N., 1992. Free rising spheres do not obey Newton's law for free settling. AIChE J. 38, 1843–1846. doi:10.1002/aic.690381116

Keffer, D., 1999. Advance numerical techniques for the solution of single nonlinear algebraic

equations and systems of nonlinear algebraic equations [WWW Document]. Univ. Tennessee. URL http://utkstair.org/clausius/docs/che505/pdf/ae_1&n_nl_n.pdf

King, M.B., 1969. Phase equilibrium in mixtures, International series of monographs in chemical engineering. Pergamon Press.

Kouba, G.E., Shoham, O., Shirazi, S., 1995. Design and performance of gas liquid cylindrical cyclone separators, in: BHR Group 7th International Conference on "Multiphase 95." Cannes, p. 23.

Kurokawa, J., Ohtaki, T., 1995. Gas-Liquid Flow Characteristics and Gas-Separation Efficiency in a Cyclone Separator.

Laleh, A.P., 2010. CFD Simulation of Multiphase Separators. ProQuest Diss. Theses. University of Calgary (Canada), Ann Arbor.

Laleh, A.P., Svrcek, W.Y., Monnery, W.D., 2012. Design and CFD studies of multiphase separators-a review. Can. J. Chem. Eng. 90, 1547–1561. doi:10.1002/cjce.20665

Langborne, P.L., 1979. Hydraulic air compression: old invention - new energy source? Chart. Mech. Eng. 26, 76–81.

Lemmon, E.W., Huber, M.L., McLinden, M.O., 2013. NIST Reference Fluid Thermodynamic and Transport Properties - REFPROP.

Mantilla, I., Shirazi, S.A., Shoham, O., 1999. Flow Field Prediction and Bubble Trajectory Model in Gas-Liquid Cylindrical Cyclone (GLCC) Separators. J. Energy Resour. Technol.

121, 9. doi:10.1115/1.2795063

McNair, F.W., Koenig, G.A., 1911. Candle tests of air from a hydraulic air compressor. Compress. Air Mag. 5963–5965.

McPherson, M.J., 2009. Subsurface Ventilation and Environmental Engineering, 2nd ed. Mine Ventilation Services Inc., Fresno.

Millar, D.L., 2014. A review of the case for modern-day adoption of hydraulic air compressors. Appl. Therm. Eng. 69, 55–77. doi:10.1016/j.applthermaleng.2014.04.008

Millar, D.L., Trapani, K., Pavese, V., 2016. WaveHAC: A Novel WEC With Power Take Off By Means Of Hydraulic Air Compressor, in: Offshore Energy and Storage Symposium. Valletta, p. 8.

Moore, L., Simonton, J., Weiler, J., 1982. The Ragged Chute Compressed Air Plant: Planning For Its Future.

Nedeltchev, S., Schumpe, A., 2011. Mass Transfer in Multiphase Systems and its Applications, in: El-Amin, M. (Ed.), Mass Transfer in Multiphase Systems and Its Applications. InTech, pp. 389–432. doi:10.5772/594

Parks Canada, 2014. Personal Communication.

Pavese, V., 2015. Energy and exergy analysis of a hydraulic air compressor for mining application. Politecnico di Torino.

Pavese, V., Millar, D.L., Verda, V., 2016. Mechanical efficiency of hydraulic air compressors. J.

Energy Resour. Technol. 138, 62005-1-62005–11. doi:10.1115/1.4033623

Perry, R.H., Green, D.W., 1999. Perry's Chemical Engineers' Handbook, 7th ed. McGraw-Hill.

Qiao, S., Mena, D., Kim, S., 2016. Inlet effects on vertical-downward air–water two-phase flow. Nucl. Eng. Des. doi:10.1016/j.nucengdes.2016.04.033

Ratcliff, G.A., Holdcroft, J.G., 1963. Diffusivities of gases in aqueous electrolyte solutions. Trans. Inst. Chem. Eng 41, 315–319.

Rice, W., 1976. Performance of Hydraulic Gas Compressors. J. Fluids Eng. 98, 645. doi:10.1115/1.3448437

Roghair, I., Lau, Y.M., Deen, N.G., Slagter, H.M., Baltussen, M.W., Van Sint Annaland, M., Kuipers, J.A.M., 2011. On the drag force of bubbles in bubble swarms at intermediate and high Reynolds numbers. Chem. Eng. Sci. 66, 3204–3211. doi:10.1016/j.ces.2011.02.030

Rowe, P.N., Henwood, G.A., 1961. Drag forces in a hydraulic model of a fluidised bed. Trans. Inst. Chem. Eng. 39, 43–54.

Sander, R., 2015. Compilation of Henry's law constants (version 4.0) for water as solvent. Atmos. Chem. Phys. 15, 4399–4981. doi:10.5194/acp-15-4399-2015

Sander, R., 1999. Compilation of Henry's law constants for inorganic and organic species of potential importance in environmental chemistry.

Scala, F., 2013. Fluidized Bed Technologies for Near-Zero Emission Combustion and Gasification, Woodhead Publishing Series in Energy. Elsevier Science.

Schulze, L.E., 1954. Hydraulic Air Compressors. United States Department of the Interior, Washington, D.C.

Schumpe, A., 1993. The estimation of gas solubilities in salt solutions. Chem. Eng. Sci. 48, 153–158. doi:10.1016/0009-2509(93)80291-W

Skogestad, S., 2008. Chemical and Energy Process Engineering. CRC Press.

Subramanian, R., 2015. ENGR 3416 Mass transfer.

Taylor, C., 1897. Hydraulic power transmission by compressed air. Montreal.

Usui, K., 1989. Vertically Downward Two-Phase Flow, (II). J. Nucl. Sci. Technol. 26, 1013–1022. doi:10.1080/18811248.1989.9734422

Vanderzee, C.E., Nutter, J.D., 1963. Heats of solution of gaseous hydrogen chloride and hydrogen bromide in water at 25°. J. Phys. Chem. 67, 2521–2523. doi:10.1021/j100806a004

Weisenberger, S., Schumpe, A., 1996. Estimation of gas solubilities in salt solutions at temperatures from 273 K to 363 K. AIChE J. 42, 298–300. doi:10.1002/aic.690420130

Whitman, W.G., 1923. The two-film theory of gas absorption. Chem. Metall. Eng. 29, 146–148.

Wilkinson, P.M., Haringa, H., Van Dierendonck, L.L., 1994. Mass transfer and bubble size in a bubble column under pressure. Chem. Eng. Sci. 49, 1417–1427. doi:10.1016/0009-2509(93)E0022-5

Williams, D., 2016. Earth Fact Sheet [WWW Document]. NASA. URL

http://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html (accessed 9.11.16).

Young, S., Pavese, V., Hutchison, A., Rico, J., Millar, D., 2016. Interphase mass transfers in hydraulic air compressors for production of mine compressed air, in: Canello, R. (Ed.), ENERMIN2016. Gecamin, Santiago, pp. 87–97.

Zhao, B., Shen, H., Kang, Y., 2004. Development of a symmetrical spiral inlet to improve cyclone separator performance, Powder Technology. doi:10.1016/j.powtec.2004.06.001

Zheng, L., Yapa, P.D., 2002. Modeling gas dissolution in deepwater oil/gas spills. J. Mar. Syst. 31, 299–309. doi:10.1016/S0924-7963(01)00067-7

Appendices

Appendix A: Sample MATLAB Code

A.1 Script to perform a downcomer simulation

```
%This script is to test the constructor and methods of the Downcomer class

clear
clear import
clc

import hacsimulator.DuctGeometry
import hacsimulator.GasSpecies
import hacsimulator.ControlParameters
import hacsimulator.Downcomer

dcGeometry = DuctGeometry(-0.4, 2.591, 2.591, 100.83, -90, 1e-3, 2)

numSegments = 20

mDotGas = 22.7

mDotLiquid = 30590

pressure = 101325

temperature = 294.15

gasMixtures = {{'nitrogen', 'oxygen'};{'nitrogen', 'oxygen','argon'};...
    {'nitrogen', 'oxygen', 'argon', 'CO2'}};

gasMolFracs = {[0.79, 0.21];[0.78, 0.21, 0.01];...
    [0.7808,0.2095, 0.0093, 0.0004];...
    [0.75, 0.05, 0.01, 0.19]};

%controls = ControlParameters(massTransfer, psychro, refpropFlag)
controls = ControlParameters(true, false, false)

liquidComponents = hacsimulator.LiquidPhase()


gasComponents = GasSpecies.empty;
gasNames = gasMixtures{3}
molFracs = gasMolFracs{3}

for jName = 1:1:length(gasNames)
    gasComponents(jName) = GasSpecies(gasNames{jName}, molFracs(jName));
end
```

```
raggedDowncomer = Downcomer(numSegments,dcGeometry, pressure, temperature,...
    mDotLiquid, liquidComponents, mDotGas, gasComponents, controls)

disp('Solving downcomer segments')
isSolved = false;

try
    profile on
    tic;
    raggedDowncomer.solveSegments();
    elapsedTime = toc
    profile off
    profsave
    isSolved = true;
    display(raggedDowncomer, 'downcomerTest')
    outletO2MolFrac = raggedDowncomer.getOutletMolFraction('oxygen',false);
    display(outletO2MolFrac, 'outletO2MolFrac')
catch
    disp('%d => NOOOOPE',iSegment)
    profile off
end
```

A.2 NewtonRaphson Class

```
classdef (Abstract) NewtonRaphson < hacsimulator.MasterHelp
    %NEWTONRAPHSON Abstract class to enable subclasses to use the Newton-
Raphson numerical solver
    %   NEWTONRAPHSON Contains the Newton-Raphson alogrithm for numerically
    %   solving a system of non-linear equations. The function that outputs
    %   the residuals is abstract and is to be implemented by NewtonRaphson's
    %   subclasses. The NewtonRaphson solver is set with a tolerance of
    %   1e-8 and a maximun number of iterations of 100.

    properties (Access = private)

        %boolean - display values of relevant parameters each iteration if
true, otherwise display nothing
        trace;

        %boolean - display converged number of iterations, RMS of residuals,
RMS of deltaX if true, otherwise display nothing
        printFinal;

    end %private properties

    properties (Constant, Access = private)

        MAX_ITERATIONS = 100; %int - the maximum iterations for numerical
procedure
        TOLERANCE = 1e-8; %double - the required tolerance for a converged
solution
```

```
end %constant properties

methods %public

    %CONSTRUCTOR
    function obj = NewtonRaphson()
        %NEWTONRAPHSON Sets the values of the properties

        %trace is set to false by default
        obj.trace = false;

        %printFinal is set to false by default
        obj.printFinal = false;

    end %CONSTRUCTOR

    function solution = solveSystem(obj, initialGuess)
        %SOLVESYSTEM Implements the multivariate Newton-Raphson method
        %   SOLVESYTEM implements the Newton-Raphson method for solving
        %   a system of non-linear equations with numerical
        %   approximations of the partial derivatives based upon the
        %   algorithm available at:
        %   http://utkstair.org/clausius/docs/che505/pdf/ae_1&n_nl_n.pdf.
        %
        %   Convergence is determined by the RMS of the deltaX vector.
        %   If the RMS of deltaX is less than the tolerance, the
        %   solution is deemed to have converged because furthur
        %   iterations would not significantly change the 'answer'.
        %
        %CALL: solution = obj.solveSystem(initialGuess)
        %
        %INPUTS:
        %   (double[]) initialGuess - initial guess of the values of
        %       the solution variables
        %
        %OUTPUTS:
        %   (double[]) solution - vector containing converged values of
        %       the solution variables

        %Determine the size of the system
        sysSize = length(initialGuess);

        %Setup parameters

        %xFlex - stores flexed x-vaules for to evaluate function
        xFlex = zeros(1, sysSize);

        %stepSize - determines how much a solution variable should be
        %flexed
        stepSize = zeros(1, sysSize);

        %jacob - stores values of numerical partial derivatives
        jacob = zeros(sysSize, sysSize);
```

```
            %fGrad - stores function values to evaluate numerical derivative
            fGrad = zeros(sysSize,sysSize,4);

            %xOld - stores values of solution parameters from previous
            %iteration for trace
            xOld = zeros(1, sysSize);

            %solution - stores current guess, initialized with initial guess
(duh doy)
            solution = initialGuess;

            %err - 'error' of the current solution used to judge
            %convergence evaluated by the RMS of deltaX. initialised to an
            %arbitrarily high number to enter while loop.
            err = 1e+10;

            %iter - current iteration checked against MAX_ITERATIONS to
prevent
            %infinite loop
            iter = 0;

            %CONVERGENCE LOOP
            while (err > obj.TOLERANCE)

                %determine step size, h, using stepSize
                for iUnkown = 1:sysSize
                    stepSize(iUnkown) = min(0.01*solution(iUnkown),0.01);
                end %stepSize loop

                %evaluate function
                residual = obj.evalResiduals(solution);
                residual = residual';

                xEval = zeros(1, sysSize);

                %set up a grid for evaluation of numerical partial
                %derivatives evaluate function at grid points
                for iUnkown = 1:1:sysSize %determines the x component to flex

                    %reset xEval to solution
                    xEval = solution;


                    %flexing the current guess in solution
                    xFlex(1) = solution(iUnkown) - 2*stepSize(iUnkown);
                    xFlex(2) = solution(iUnkown) - stepSize(iUnkown);
                    xFlex(3) = solution(iUnkown) + stepSize(iUnkown);
                    xFlex(4) = solution(iUnkown) + 2*stepSize(iUnkown);

                    for jEvaluation = 1:1:4
                        %flexes solution(j) for function evaluation
                        xEval(iUnkown) = xFlex(jEvaluation);
```

```
                        %evaluate function values required for all partial
                        %derivatives with respect to solution(j)
                        fGrad(:,iUnkown,jEvaluation) =
obj.evalResiduals(xEval);

                    end %jEvaluation loop

                end %iUnknown loop

                %calculate jacobian matrix
                for iEquation = 1:1:sysSize
                    for jUnkown = 1:1:sysSize
                        %evaluate numerical derivative
                        jacob(iEquation,jUnkown) = ...
                            (-fGrad(iEquation,jUnkown,4) + ...
                            8*fGrad(iEquation,jUnkown,3) - ...
                            8*fGrad(iEquation,jUnkown,2) + ...
                            fGrad(iEquation,jUnkown,1))/...
                            (12*stepSize(jUnkown));

                    end %jUnknown loop
                end %iEquation loop

                %eval deltaX
                deltaX = jacob\residual;

                %evaluate new estimate of solution
                for iUnkown = 1:1:sysSize
                    xOld(iUnkown) = solution(iUnkown); %used for tracing
purposes
                    solution(iUnkown) = solution(iUnkown) - deltaX(iUnkown);
                end %new estimate loop

                %check for convergence
                err = sqrt(sum(deltaX.^2)/sysSize);


                if obj.trace

                    NUM_SIG_FIGS = 8;
                    display(iter)
                    display(num2str(residual,NUM_SIG_FIGS),'Residuals')
                    rmsResiduals = sqrt(sum(residual.^2)/sysSize);
                    display(num2str(rmsResiduals, NUM_SIG_FIGS),'RMS
Residual')
                    display(num2str(jacob,NUM_SIG_FIGS),'Jacobian Matrix')
                    display(num2str(deltaX,NUM_SIG_FIGS),'deltaX')
                    display(num2str(err,NUM_SIG_FIGS), 'error')
                    display(num2str(xOld,NUM_SIG_FIGS),'xOld')
                    display(num2str(solution,NUM_SIG_FIGS), 'solution')

                end %trace check
```

```
                iter = iter + 1;

                if iter > obj.MAX_ITERATIONS
                    display(num2str(residual,5),'residual')

error('NewtonRaphson:solveSystem:maxIterationsExceeded',...
                        'Maximum number of iterations exceeded');
                end %if

            end %while loop

            if obj.printFinal

                NUM_SIG_FIGS = 8;
                display(iter)
                display(num2str(solution,NUM_SIG_FIGS), 'solution')
                display(num2str(err, NUM_SIG_FIGS),'Error')
                rmsResiduals = sqrt(sum(residual.^2)/sysSize);
                display(num2str(rmsResiduals, NUM_SIG_FIGS),'RMS Residuals')

            end %printFinal check


        end %solveSystem

        function switchSolverTrace(obj)
            %SWITCHSOLVERTRACE Switches trace from false to true and vice
versa
            %   SWITCHSOLVERTRACE Switches trace so that intermediate
            %   values of Newton-Raphson algoritm will or will not be
            %   displayed
            %
            %CALL: obj.switchSolverTrace()

            obj.trace = ~obj.trace;

        end %turnSolverTraceOn

        function switchPrintFinal(obj)
            %SWITCHPRINTFINAL Switches printFinal from false to true and vice
versa
            %   SWITCHPRINTFINAL Switches printFinal so final values of
            %   iter, RMS of residuals, RMS of deltaX will or will not be
            %   displayed.
            %
            %CALL: obj.switchPrintFinal()

            obj.printFinal = ~obj.printFinal;

        end %switchPrintFinal
```

```
    end %public methods

    methods (Access = protected, Abstract)


        resid = evalResiduals(obj, currentGuess)
        %EVALRESIDUALS Evaluates the residuals of the system of equations
        %   EVALRESIDUALS Abstract method to be implemented by subclasses
        %   which outputs residuals of the system of equations to be solved
        %
        %CALL: resid = obj.evalResiduals(currentGuess)
        %
        %INPUTS:
        %   (double[]) currentGuess - current guess of the solution of the
        %       system
        %
        %OUTPUTS:
        %   (double[]) resid - residuals of the system of equations

    end %abstract methods

end %NewtonRaphson class
```

A.3 Downcomer

```
classdef Downcomer < hacsimulator.NewtonRaphson
    %DOWNCOMER Evaluates and stores the solution of the downcomer
    %   This class guides the solution process for determining the state
    %   (pressure, temperature, water velocity, gas slip veloctiy and gas
    %   composition) of of the fluids at the outlet of the downcomer. The
    %   results of the solution are stored in an array of DowncomerSegment
    %   objects, the power losses are accumulated, and the
    %   root-mean-square of the conservation residuals (energy, momentum,
    %   mass, slip velocity, species) are determined.
    %
    %   Using the DowncomerSegment objects allows for either a verbose or a
    %   succinct solution can be attained by querying the desiered object
    %   properties. The solution process also no longer needs the 'packing'
    %   and 'unpacking' of a 'knowns' array that can exceed 26 elements.
    %
    %Author: Stephen Young, MIRARCO, Sudbury (syoung@mirarco.org)
    %Date: April 2016

    properties (SetAccess = protected)

        segments; %DowncomerSegment[] - Array of DowncomerSegment objects
        numSegments; %int - The number of segments used
        species; %String{} - cell array with names of species under
consideration

        %Losses
```

```
        bubbleDragLoss; %double - total power loss from bubble drag in
downcomer [W]
        wallFrictionLoss; %double - total power loss from wall friction in
downcomer [W]

        %Residuals

        rmsEnergy; %double - RMS of the energy conservation residuals
        rmsMomentum; %double - RMS of the momentum conservation residuals
        rmsMass; %double - RMS of the mass conservation residuals
        rmsSlipVelocity; %double - RMS of the slip velocity equation
residuals
        rmsSpecies; %double[] - array containing the RMS of the species
conservation residual for each species

    end %gettable properties

    properties (Access = private)

        segPosition; %int - index of the segment that is currently being
solved
        massTransfer; %boolean - include mass transfer if TRUE, otherwise
exclude
        twoStage; %boolean - use two stage solution procedure if TRUE,
otherwise single stage

    end %private properties

    methods

        %CONSTRUCTOR
        function obj = Downcomer(numSegments, geometry, pressure,
temperature,...
                mDotLiquid, liquidComponents, mDotGas, gasComponents,
controls)
        %DOWNCOMER Constructs the Downcomer object and sets the values of the
properties.
        %
        %CALL: obj = Downcomer(numSegments, geometry, mDotLiquid, mDotGas,...
        %        pressure, temperature, gasComponents, controls)
        %
        %INPUTS:
        %   (int) numSegments - the number of segments to divide the
        %       downcomer into
        %   (DuctGeometry) geometry - DuctGeometry object for the downcomer
        %   (double) pressure - pressure of fluids at downcomer inlet [Pa]
        %   (double) temperature - temperature of fluids at downcomer [K]
        %   (double) mDotLiquid - mass flow rate of the liquid phase
        %   (LiquidPhase) liquidComponents - LiquidPhase describing key
        %       parameters of the liquid phase.
        %   (double) mDotGas - mass flow rate of the gas phase
        %   (GasSpecies[]) gasComponents - GasSpecies array with names
        %       and mole fractions of gas species included in analysis
        %   (ControlParameters) controls - ControlParameters object with
```

```
        %       values of solution controls

            %Call super class constructor
            obj@hacsimulator.NewtonRaphson();

            errorFound = false;

            %input checking extravaganza
            if ~hacsimulator.NumberCheck.isNumber(numSegments)
                errorFound = true;
                errorMessage = 'numSegments should be a real scalar integer';
            elseif ~isa(geometry, 'hacsimulator.DuctGeometry')
                errorFound = true;
                errorMessage = 'geometry must be of type DuctGeometry';
            elseif ~hacsimulator.NumberCheck.isNumber(mDotLiquid)
                errorFound = true;
                errorMessage = 'mDotLiquid must be a real scalar number';
            elseif ~hacsimulator.NumberCheck.isNumber(mDotGas)
                errorFound = true;
                errorMessage = 'mDotGas must be a real scalar number';
            elseif ~hacsimulator.NumberCheck.isNumber(pressure)
                errorFound = true;
                errorMessage = 'pressure must be a real scalar number';
            elseif ~hacsimulator.NumberCheck.isNumber(temperature)
                errorFound = true;
                errorMessage = 'temperature must be a real scalar number';
            elseif ~isa(gasComponents, 'hacsimulator.GasSpecies')
                errorFound = true;
                errorMessage = 'gasComponents must be of type GasSpecies';
            elseif ~isa(liquidComponents, 'hacsimulator.LiquidPhase')
                errorFound = true;
                errorMessage = 'liquidComponents must be of type
LiquidPhase';
            elseif ~isa(controls, 'hacsimulator.ControlParameters') ||
~isscalar(controls)
                errorFound = true;
                errorMessage = 'controls must be a scalar of type
ControlParameters';
            end %input checking

            if errorFound
               error('Downcomer:Downcomer:illegalArgument',errorMessage)
            end %error check

            %set numSegments & speices
            obj.numSegments = numSegments;


            %collect relevant control parameters
            obj.massTransfer = controls.massTransfer;
            obj.twoStage = controls.twoStage;

            %initialise losses
            obj.bubbleDragLoss = 0;
```

```
            obj.wallFrictionLoss = 0;

            %initialise residuals
            obj.rmsEnergy = 0;
            obj.rmsMomentum = 0;
            obj.rmsMass = 0;
            obj.rmsSlipVelocity = 0;
            obj.rmsSpecies = zeros(1,length(gasComponents));

            %Initialise the first DowncomerSegment

            %construct DowncomerInitialiser
            initialiser = hacsimulator.DowncomerInitialiser(geometry, ...
                pressure, temperature, mDotLiquid, liquidComponents,
mDotGas,...
                gasComponents, controls);

            inState = [pressure, temperature, initialiser.liquidVel, ...
                initialiser.slipVel];

            %set up segment geometry
            segmentGeometry = geometry.evalSegmentGeometry(numSegments);

            %construct first segment of segments array
            obj.segments = hacsimulator.DowncomerSegment(segmentGeometry,...
                inState, mDotLiquid, liquidComponents, mDotGas, ...
                initialiser.gasMixture, initialiser.bubbleFlux, controls);

            %Initialise segPosition
            obj.segPosition = 1;

            %set species property
            obj.species = initialiser.gasMixture.getNameArray(false);

        end %constructor

        function solveSegments(obj)
            %SOLVESEGMENTS Computes outlet state (pressure, temperature,
water velocity,
            %gas slip velocity, and gas composition)of the downcomer from
initial
            %conditions at the inlet.
            %   This function implements the abstract NewtonRaphson class
            %   and DowncomerSegment objects to determine the pressure,
            %   temperature, water velocity, gas relative velocity and gas
            %   composition at the outlet of a downcomer pipe by dividing
            %   it into segments and forward stepping through the downcomer
            %   maintaining the conservation of mass, momentum and energy
            %   and gas species in each segment.
            %
            %CALL: obj.solveSegments()

            %{
```

```
                V1  ->  converted VBA Code
                V2  ->  Uses a DowncomerData object instead of the ordered
knowns
                        vector
                    ->  Calls downcomerSegmentV2 and dowcomerSegmentV3
                        rather than downcomerSegment
                    ->  Includes solubility and psychrometric aspects
                V3  ->  Converted to a method in the object Downcomer
                    ->  Uses object functionality from NewtonRaphson and
                        DowncomerSegment
            %}

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            %initialise resid temp variables
            energySumOfSquares = 0;
            momentumSumOfSquares = 0;
            massSumOfSquares = 0;
            slipVelSumOfSquares = 0;

            if obj.massTransfer
                speciesSumOfSquares = zeros(1,length(obj.species));
            end

            %CALCULATION LOOP TO GO THROUGH SEGMENTS
            for iSegment = 1:1:obj.numSegments

                %obj.segments(1) is constructed in this.constructor

                if iSegment ~= 1
                    %Construct a DowncomerData object for the current
                    %segment from previous segment
                    obj.segments(iSegment) =
hacsimulator.DowncomerSegment(...
                        obj.segments(iSegment-1));
                end

                initialGuess =
obj.segments(iSegment).outlet.getStateVector();

                %refine initial guess of outlet state assuming constant fluid
                %properties and no interphase mass transfer

                initialGuess = obj.solveSystem(initialGuess);

                %Update state at segment outlet.
                obj.segments(iSegment).updateOutletState(initialGuess);

                %second stage solution doesn't assume constant fluid
proerties in the segment
                if obj.twoStage

                    %set firstStage flag to false
```

```
                    obj.segments(iSegment).secondStage();

                    %get initial guess including molar flowrates
                    initialGuess =
obj.segments(iSegment).outlet.getStateVector();

                    %refine solution assuming linear variation of fluid
                    %properties and include interphase mass transfers
                    %(depending upon controlParameters)
                    initialGuess = obj.solveSystem(initialGuess);

                    %Update outlet state
                    obj.segments(iSegment).updateOutletState(initialGuess)

                end %twoStage if

                %evaluate bubble drag and wall friction losses
                obj.segments(iSegment).setLosses();

                %retrieve residuals of solution from donwncomerSegment
                resid = obj.segments(iSegment).evalResiduals(initialGuess);

                %accumulate residuals

                energySumOfSquares = energySumOfSquares + resid(1)^2;
                momentumSumOfSquares = momentumSumOfSquares + resid(2)^2;
                massSumOfSquares = massSumOfSquares + resid(3)^2;
                slipVelSumOfSquares = slipVelSumOfSquares + resid(4)^2;

                if obj.massTransfer

                    speciesSumOfSquares = speciesSumOfSquares +
resid(5:end).^2;

                end %species residual accumulation

                %increment segPosition
                obj.segPosition = obj.segPosition + 1;

            end %main calculation loop

            %accumulate losses in Downcomer
            obj.accumulateLosses()

            %evaluating rms of each residual
            obj.rmsEnergy = sqrt(energySumOfSquares/double(obj.numSegments));
            obj.rmsMass = sqrt(massSumOfSquares/double(obj.numSegments));
            obj.rmsMomentum =
sqrt(momentumSumOfSquares/double(obj.numSegments));
            obj.rmsSlipVelocity =
sqrt(slipVelSumOfSquares/double(obj.numSegments));
```

```
            if obj.massTransfer
                obj.rmsSpecies = sqrt(speciesSumOfSquares ./
double(obj.numSegments));
            end %massTransfer check

            %{
            % disp('Segments solved!')
            %
display(num2str(obj.segments(obj.numSegments).outlet.getStateVector(false),..
.
            %     5),'downcomerOutletState')
            %}

        end %solveSegments

        function position = getSegPosition(obj)
            position = obj.segPosition;
        end

        function molFracProfile = getMixtureMolFracProfile(obj, humid)
            %GETMOLFRACTIONPROFILE Gets the mol fraction profile of each
            %species at the outlet of each DowncomerSegment in segments
            %
            %CALL: molFracProfile = obj.getMolFracArrayProfile(humid)
            %
            %INPUTS:
            %   (boolean) humid - get humid mole fractionss if true,
            %       otherwise get dry mole fractions
            %
            %OUTPUTS:
            %   (double[][]) molFracProfile - array with mol fractions of
            %       each species down the downcomer


            %molFracProfile preallocation
            molFracProfile = zeros(obj.numSegments,length(obj.species));

            for iSegment = 1:1:obj.numSegments
                molFracProfile(iSegment, :) = ...
                    obj.segments(iSegment).outlet.mixture.getMolFracArray(...
                    humid);
            end %retrieval loop

        end %getMolFracArrayProfile

        function molFracProfile = getMolFracProfile(obj, name, humid)
            %GETMOLFRACTIONPROFILE Gets the mol fraction profile of each
            %species at the outlet of each DowncomerSegment in segments
            %
            %CALL: molFracProfile = obj.getMolFracProfile(name, humid)
            %
            %INPUTS:
            %   (String) name - name of the given speices
            %   (boolean) humid - get humid mole fractionss if true,
```

```
            %       otherwise get dry mole fractions
            %
            %OUTPUTS:
            %   (double[]) molFracProfile - array with mol fractions of
            %       a given species down the downcomer


            %molFracProfile preallocation
            molFracProfile = zeros(obj.numSegments,1);

            for iSegment = 1:obj.numSegments
                molFracProfile(iSegment) =
obj.segments(iSegment).outlet.mixture...
                    .getMolFrac(name, humid);
            end %retrieval loop

        end %getMolFracProfile

        function outletMolFrac = getOutletMolFraction(obj, name, humid)
            %GETOUTLETMOLFRACTION Gets the mole fraction of a given species
            %at the outlet of the downcomer.
            %
            %CALL: outletMolFrac = obj.getOutletMolFraction(name)
            %
            %INPUTS:
            %   (String) name - name of species the outlet mol fraction is
            %       desired
            %   (boolean) humid - get humid mol fraction if true, otherwise
            %       get dry mol fraction
            %
            %OUTPUTS:
            %   (double) outletMolFrac - mole fraction of species, name, at
            %       downcomer outlet

            outletMolFrac = obj.segments(end).outlet.mixture...
                .getMolFrac(name, humid);

        end %getOutletMolFraction

        function engConcProfile = evalEngConcProfile(obj)
            %EVALENGCONCPROFILE Evaluates the engineering concentration
            %profile of gas solutes in the liquid phase down the downcomer
            %
            %CALL: engConcProfile = obj.evalEngConcProfile()
            %
            %OUTPUTS:
            %   (double[][]) engConcProfile - engineering concetnration of
            %       gas solute [g gas/ kg water]

            engConcProfile = zeros(obj.numSegments, length(obj.species));

            %calculation loop
            for iSegment = 1:obj.numSegments
```

```
                for jSpecies = 1:1:length(obj.species)
                    KG_TO_G = 1000;

                    %eval bulk concentration
                    bulkConc = obj.segments(iSegment).outlet.evalBulkConc(...
                        obj.species{jSpecies});

                    %get liquid phase density
                    liquidDensity = obj.segments(iSegment).outlet.rhoLiquid;

                    %get molar mass of species
                    molarMass = obj.segments(iSegment).outlet.mixture...
                        .getMolMass(obj.species{jSpecies});

                    %eval engineering concentratoin [g gas / kg water]
                    engConcProfile(iSegment,jSpecies) =
bulkConc*molarMass*KG_TO_G/...
                         liquidDensity;
                end %species loop

            end %segment loop


        end %getEngConcentrationProfile

    end %public methods

    methods (Access = protected)

        function accumulateLosses(obj)
            %ACCUMULATELOSSES Sets bubbleDragLoss and wallFrictionLoss
            %   ACCUMULATELOSSES Determines the total power loss from wall
            %   firction and interfacial drag by summing the current values
            %   of bubbleDragLoss and wallLoss in the array of
            %   DowncomerSegment objects
            %
            %CALL: obj.accumulateLosses()

            for iSegment = 1:obj.numSegments

                %bubble drag
                obj.bubbleDragLoss = obj.bubbleDragLoss + ...
                    obj.segments(iSegment).bubbleDragLoss;

                %wall friction
                obj.wallFrictionLoss = obj.wallFrictionLoss + ...
                    obj.segments(iSegment).wallLoss;

            end %for loop

        end %accumulateLosses
```

```
        function resid = evalResiduals(obj, currentGuess)
            %EVALRESIDUALS Evaluates the residuals of the conservation
equations of the current segment being solved
            %   EVALRESIDUALS Evaluates the residuals of the conservation
            %   of energy, momentum, mass, slip velocity, and species using
            %   the evalResiduals method of the current segment being
            %   solved based on the current guess for pressure,
            %   temperature, water velocity, gas slip velocity, and molar
            %   flowrate of each species in the gas phase
            %
            %CALL: resid = obj.evalResiduals(currentGuess)
            %
            %INPUTS:
            %   (double[]) currentGuess - current guess of solution
            %       variables: pressure, temperature, water velocity, gas
            %       slip velocity, and molar flowrate of each species in
            %       the gas phase
            %
            %OUTPUTS:
            %   (double) resid - residuals of energy, momentum, mass, slip
            %       velocity and species conservation equations

            resid =
obj.segments(obj.segPosition).evalResiduals(currentGuess);

        end %evalResiduals

    end %private methods

end
```

A.4 DowncomerSegment

```
classdef DowncomerSegment < hacsimulator.MasterHelp
    %DOWNCOMERSEGMENT Contains required properties and methods for a
downcomer segment.
    %   The DOWNCOMERSEGMENT class contains the properties and methods
    %   necessary to evaluate and store the properties required to
    %   determine the residuals of the conservation equations (energy,
    %   momentum, mass, species, and slip velocity) and evaluate residuals
    %   themselves in a segment of the downcomer flow in a hydraulic air
    %   compressor.
    %
    %Author: Stephen Young, MIRARCO, Sudbury (syoung@mirarco.org)
    %Date: April 2016


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    properties (SetAccess = protected)
        %outlet conditions

        outlet; %DowncomerSection object for segment outlet
```

```
        %inlet conditions

        inlet; %DowncomerSection object for segment inlet

        %duct geometry
        geometry; %SegmentGeometry - SegmentGeometry object for this segment

        %losses

        wallShear; %double - wall shear stress from fluid flow [Pa]
        wallLoss; %double - power loss due to fluid frction with wall [W]
        bubbleDragLoss; %double - power loss to to interfacial drag [W]

        %two-phase flow parameters

        numBubbles; %double - number of bubbles in the segment
        gasVolFrac; %double - volume fraction of gas in the segment
        avgBubbleDia; %double - average bubble diameter in the segment [m]

    end %gettable properties

    properties (Access = private)

        liquidComponents; %LiquidPhase - object describing key parameters of
liquid phase
        gases; %String{} - cell array with the names of the dry gaseous
species in analysis
        controls; %ControlParameters object handle from Downcomer
        firstStage = true; %boolean - TRUE - in first stage of solution
process, FALSE - in second stage

    end %private properties


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    methods

        %CONSTRUCTOR
        function obj = DowncomerSegment(varargin)
            %Constructs the object and initializes all properties
            %
            %Case nargin = 1:
            %
            %CALL: obj = DowncomerSegment(previousSeg)
            %
            %INPUTS:
            %   (DowncomerSegment) previousSeg - the DowncomerSegment
            %       object for the previously solved segment
            %
            %Case nargin = 8:
            %
```

```
%CALL: obj = DowncomerSegment(geometry, inState, mDotLiquid, ...
%    liquidComponents, mDotGas , gasMixture, bubbleFlux,...
%    controls)
%
%INPUTS:
%    (SegmentGeometry) geometry - SegmentGeometry object for
%        the segment
%    (double[]) inState -  vector with pressure [Pa],
%        temperature [K], velocity of liquid phase [m/s],
%        and slip velocity of gas phase [m/s].
%    (double) mDotLiquid - mass flow rate of liquid phase
%        at segment inlet
%    (LiquidPhase) liquidComponents - LiquidPhase object with
%        names of bulk fluid and cosolute in liquid phase and
%        cosolute concentration
%    (double) mDotGas - mass flow rate of gas phase at
%        segment inlet
%    (SpeciesTracker) gasMixture - SpeciesTracker object
%        for segment inlet
%    (double) bubbleFlux - bubble flux for downcomer segment
%    (ControlParameters) controls - ControlParameters object
%        with values of the solution controls

obj.segInputCheck(varargin);

switch nargin

    case 1

        %collect input from varargin
        previousSeg = varargin{1};

        %pass the prevous segment's outlet handle to obj.inlet
        obj.inlet = previousSeg.outlet;

        %control parameters
        obj.controls = previousSeg.controls;

        %get LiquidPhase object handle
        obj.liquidComponents = previousSeg.liquidComponents;

        %evaluate geometry of this segment
        obj.geometry = previousSeg.geometry.evalNextSegment();

        %get values to initialise outlet section
        state = [previousSeg.outlet.pressure,...
            previousSeg.outlet.temp,...
            previousSeg.outlet.liquidVel, ...
            previousSeg.outlet.slipVel];

        gasMixture = previousSeg.outlet.mixture;
        outletDia = obj.geometry.outletDiameter;
        outletFlowArea = obj.geometry.evalOutletFlowArea();
```

```
            outletEL = obj.geometry.evalOutletElevation();
            mDotLiquid = previousSeg.outlet.mDotLiquid;
            mDotGas = previousSeg.outlet.mDotGas;
            bubbleFlux = previousSeg.outlet.bubbleFlux;

            %construct outlet DowncomerSection object
            obj.outlet = hacsimulator.DowncomerSection(outletDia, ...
                outletFlowArea, outletEL, state, mDotLiquid, ...
                obj.liquidComponents, mDotGas, gasMixture, ...
                bubbleFlux, obj.controls);


        case 8

            %collect variables from varargin
            geometry = varargin{1};
            inState = varargin{2};
            mDotLiquid = varargin{3};
            liquidComponents = varargin{4};
            mDotGas = varargin{5};
            gasMixture = varargin{6};
            bubbleFlux = varargin{7};
            controls = varargin{8};

            %controls
            obj.controls = controls;

            %get LiquidPhase object
            obj.liquidComponents = liquidComponents;

            %segment geometry
            obj.geometry = geometry.deepCopy();

            %inlet geometry
            inletEL = geometry.startElevation;
            inletDia = geometry.inletDiameter;
            inletFlowArea = geometry.evalInletFlowArea();

            %outlet geometry
            outletEL = geometry.evalOutletElevation();
            outletDia = geometry.outletDiameter;
            outletFlowArea = geometry.evalOutletFlowArea();

            %inlet and outlet conditions

            obj.inlet = hacsimulator.DowncomerSection(inletDia, ...
                inletFlowArea, inletEL, inState, mDotLiquid,...
                liquidComponents, mDotGas, gasMixture,
bubbleFlux,...
                obj.controls);

            %outlet initialised with inlet conditions
            obj.outlet = hacsimulator.DowncomerSection(outletDia, ...
```

```
                        outletFlowArea, outletEL, inState, mDotLiquid, ...
                        liquidComponents, mDotGas, gasMixture, bubbleFlux,
...
                        obj.controls);

            end %nargin switch

            obj.gases = obj.inlet.mixture.getNameArray(false);



            %set two phase flow parameters
            obj.setAvgBubbleDiameter();
            obj.setGasVolumeFraction();
            obj.setNumBubbles();

            %Initialise losses and wallShear to zero
            obj.wallLoss = 0;
            obj.bubbleDragLoss = 0;
            obj.wallShear = 0;



        end %CONSTRUCTOR

        function secondStage(obj)
            %SECONDSTAGE Sets control parameter firstStage to false for
second stage solution
            %   SECONDSTAGE Sets the property firstStage to false so that
            %   method calls for the DowncomerSegment object can include
            %   solubility, psychrometry, and variable fluid properties
            %
            %CALL: obj.secondStage()

            obj.firstStage = false;
            obj.inlet.secondStage();
            obj.outlet.secondStage();

        end %secondStage

        function setLosses(obj)
            %SETLOSSES Setter method for the wallLoss and bubbleDragLoss
properties
            %   SETLOSSES Determines the losses in the downcomer segment
            %   due to fluid friction on the walls and interfacial fluid
            %   drag on the bubbles and assigns the values to the
            %   properties wallLosss and bubbleDragLoss.
            %
            %CALL: obj.setLosses()

            %BUBBLE DRAG POWER LOSS
```

```
            %check bubbleDrag flag
            if obj.controls.bubbleDrag
                %determine drag coefficient based on particle Reynolds number
                dragCoeff = obj.evalDragCoefficient();
            else
                %drag not included in analysis -> set dragCoeff to zero
                dragCoeff = 0;
            end
            %bubble drag power loss bubbleLoss =
numBubbles*bubbleDrag*slipVelOut [W]
            obj.bubbleDragLoss = obj.numBubbles * ...
                (1/8) * dragCoeff * pi * obj.avgBubbleDia^2 * ...
                obj.outlet.rhoLiquid * obj.outlet.slipVel^3;

            %WALL fRICTION POWER LOSS

            %determine average velocity
            if obj.controls.wallFriction == 0 %fictionless

                avgVel = 0;

            elseif obj.controls.wallFriction == 1 %two-phase

                %total mass flowrate
                mDotTotal = obj.inlet.mDotGas + obj.inlet.mDotLiquid;

                %mass flowrate weighted average of the density of both
                %phases
                avgRho =
(obj.inlet.mDotLiquid/mDotTotal)*(obj.inlet.rhoLiquid + ...
                    obj.outlet.rhoLiquid)/2 +
(obj.inlet.mDotGas/mDotTotal)*...
                    (obj.inlet.rhoGas + obj.outlet.rhoGas)/2;

                %average velocity = (total mass flowrate) / (average density)
* (outlet area)
                avgVel = mDotTotal/(avgRho*obj.outlet.area);

            else %water phase only

                avgVel = 0.5*(obj.inlet.liquidVel + obj.outlet.liquidVel);

            end %wallFriction check

            %wall power loss = wallShear * avgVel * wallArea
            obj.wallLoss = obj.wallShear * avgVel* obj.geometry.wallArea;

        end %setLosses

        function updateOutletState(obj, newState)
            %UPDATEOUTLETSTATE updates the state variables of the segment's
            %outle, molar flowrates of each species in both phases, and the
            %mass flowrates of each phase.
```

```
            %
            %CALL: obj.updateOutletState(newState)
            %
            %INPUTS:
            %   (double[]) newState - vector with the new guess for
            %       pressure, temperature, water velocity, gas slip
            %       velocity, and the molar flowrates of species in the
            %       gaseous phase

            obj.outlet.updateState(newState)

            %interphase mass transfer only considered in second stage

            if ~obj.firstStage &&...
                    (obj.controls.massTransfer || obj.controls.psychro)
                %get mass transfer. positive when species are leaving gas
phase
                %and entering water phase.

                %get change in liquid molar flowrate of each species
                deltaMolFlow = obj.outlet.mixture.getNDotLiquidArray() - ...
                    obj.inlet.mixture.getNDotLiquidArray();

                %get molar mass of each species
                molarMassArray = obj.inlet.mixture.getMolMassArray();

                %get mass transfer rate due to solubility
                solubilityTransferRate = sum(deltaMolFlow .* ...
                    molarMassArray);

                %mass transfer from psychrometry
                if obj.controls.psychro
                    mDotDryIn = obj.inlet.mDotGas * ...
                        (1 - obj.inlet.mixture.getWaterMassFrac());

                    mDotDryOut = mDotDryIn - solubilityTransferRate;

                    %mass transfering from gas to liquid phase is positive
                    psychoTransferRate = mDotDryIn * ...
                        obj.inlet.mixture.absHumidity - ...
                        mDotDryOut * obj.outlet.mixture.absHumidity;
                else
                    psychoTransferRate = 0;
                end %psychro check

                massTransferRate = solubilityTransferRate +
psychoTransferRate;

                %display(solubilityTransferRate)
                %display(psychoTransferRate)
                %display(massTransferRate)
```

```
            newMDotWater = obj.inlet.mDotLiquid + massTransferRate;
            newMDotGas = obj.inlet.mDotGas - massTransferRate;

            %display(massTransferRate)

            obj.outlet.updateMassFlows(newMDotWater, newMDotGas)

        end %firstStage/massTransfer/psychro check

        %update two-phase flow properties
        obj.setAvgBubbleDiameter()


    end %updateOutletState

    function resid = evalResiduals(obj, outState)
        %EVALRESIDUALS Evaluates the residuals of the governing
conservation equations in a downcomer segment.
        %   EVALRESIDUALS evaluates the values of the residuals of the
        %   simultaneous non-linear equations:
        %               1. Energy conservation
        %               2. Momentum conservation
        %               3. Mass conservation
        %               4. Slip velocity equation
        %               5. Spieces conservation of nitrogen
        %               6. Spieces conservation of oxygen
        %
        %   Used by the Downcomer class to solve for the unknowns:
        %   pressure, temperature, water velocity, slip veloctiy of the
        %   gas, and molar flowrate of gaseous species at the segment
        %   outlet.
        %
        %   CALL: resid = obj.evalResiduals(outState)
        %
        %   INPUTS:
        %       outState - (double[]) array with provisional values of
        %           pressure, temperature, water velocity, and bubble
        %           slip velocity at the outlet
        %
        %   OUTPUTS:
        %       resid - (double[]) residuals of the solution equations
        %
        %   V1  -> Converted code from VBA function in HAC mk14
        %
        %   V2  -> Added interfacing with DowncomerData,
        %       DowncomerSection, and SpeciesTracer objects
        %       -> Added solubility of gases and species conservation
        %       equations
        %       -> Implemented drag coefficient correction with gas
        %       volume fraction.
        %       -> Changed evaulation of internal energy from Cv*T to a
        %       Refprop/CoolProp lookup
        %   V3: -> First stage of two stage solution procedure assuming
        %       no mass transfer
```

```
        %   V4  -> Combined V2 & V3 using firstStage and massTransfer
        %       flags
        %       -> Converted function to a method inside DowncomerData
        %       -> Renamed DowncomerData to DowncomerSegment

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %initialize output
        resid = zeros(1,length(outState));

        %Update values of outlet state
        obj.updateOutletState(outState)

        %first stage assumes constant fluid properties in segment
        if ~obj.firstStage
            obj.outlet.updateFluidProps()
        end %firststage if

        %Conservation of energy:
        resid(1) = obj.evalEnergyResid();

        %Conservation of momentum
        resid(2) = obj.evalMomentumResid();

        %Conservation of mass:
        resid(3) = obj.evalMassResid();

        %slip velocity residual
        resid(4) = obj.evalSlipVelocityResid();

        %SPECIES CONSERVATION

        if ~obj.firstStage && obj.controls.massTransfer

            resid(5:end) = obj.evalSpeciesResid;

        end %firstStage and massTransfer check

    end %evalResiduals

end %public methods

methods (Access = protected)

    function segInputCheck(obj, args)
        %Method to check the varargin input to the constructor
        %
        %CALL: obj.segInputCheck(args)
        %
        %INPUTS:
        %    (cell) args - varargin input to the constructor
        %
```

```
            %THROWS:
            %    illegalArgument - if an argument is found to be not of the
            %        right type

            %build error I.D. string
            errID = 'DowncomerSegment:segInputCheck:inputMismatch';

            errorDetected = false;

            numArgs = length(args);

            if numArgs == 1
                if ~isa(args{1}, 'hacsimulator.DowncomerSegment')

                    %build the error message string
                    errMsg = ['Argument should be of type Downcomer',...
                        'Segment for the nargin == 1 case'];

                    errorDetected = true;

                end %input checking

            elseif numArgs == 8

                for iArg = 1:1:numArgs
                    if iArg == 1
                        if ~isa(args{iArg},'hacsimulator.SegmentGeometry') ||
~isscalar(args{iArg})
                            errMsg = ['geometry should be of type ',...
                                'SegmentGeometry'];
                            errorDetected = true;
                            break
                        end
                    elseif iArg == 2
                        if ~isnumeric(args{iArg})
                            errMsg = ['inState should be of type ',...
                                'double'];
                            errorDetected = true;
                            break
                        end
                    elseif iArg == 4
                        if ~isa(args{iArg},'hacsimulator.LiquidPhase') ||
~isscalar(args{iArg})
                            errMsg = ['gasMixture should be a scalar of type
',...
                                'LiquidPhase'];
                            errorDetected = true;
                            break
                        end
                    elseif iArg == 6
                        if ~isa(args{iArg},'hacsimulator.SpeciesTracker') ||
~isscalar(args{iArg})
                            errMsg = ['gasMixture should be of type ',...
                                'SpeciesTracker'];
```

```
                            errorDetected = true;
                            break
                        end
                    elseif iArg == length(args)
                        if ~isa(args{iArg}, 'hacsimulator.ControlParameters')
|| ~isscalar(args{iArg})
                            errMsg = ['controls should be an object ',...
                                'of type ControlParameters'];
                            errorDetected = true;
                            break
                        end
                    else
                        if ~hacsimulator.NumberCheck.isNumber(args{iArg})
                            errMsg = ['Invalid input. varargin(', ...
                                num2str(iArg), ') should be a real scalar
',...
                                'number.'];
                            errorDetected = true;
                            break
                        end
                    end %iteration check

                end %for loop through args

            else
                errMsg = ['DowcomerSegment received the wrong number of',...
                    ' arguments.'];
                errorDetected = true;
            end

            %throw error
            if errorDetected
                error(errID, errMsg)
            end %errorDetected check

        end %segInputCheck

        function setNumBubbles(obj)
            %SETNUMBUBBLES Sets the numBubbles property
            %   setNumBubbles set the numBubbles property based upon the
            %   current values of gasVolFrac and avgBubbleDia.
            %
            %CALL: obj.setNumBubbles()

            gasVolume = obj.gasVolFrac * obj.geometry.volume;

            bubbleVolume = pi* obj.avgBubbleDia^3 / 6;

            obj.numBubbles = gasVolume / bubbleVolume;

        end %setNumBubbles

        function setAvgBubbleDiameter(obj)
```

```
            %SETAVGBUBBLEDIAMETER Setter method for avgBubbleDia.
            %
            %CALL: obj.setAvgBubbleDiameter()

            obj.avgBubbleDia = 0.5*(obj.inlet.evalBubbleDia() + ...
                obj.outlet.evalBubbleDia());

        end %setAvgBubbleDiameter

        function setGasVolumeFraction(obj)
            %SETGASVOLUMEFRACTION Setter method for gasVolFrac
            %
            %CALL: obj.setGasVolumeFraction()

            %determine superficial gas velocity at inlet
            superGasVel = obj.inlet.mDotGas / (obj.inlet.rhoGas * ...
                obj.inlet.area);

            %determine superficial liquid velocity at inlet
            superLiquidVel = obj.inlet.mDotLiquid / ...
                (obj.inlet.rhoLiquid * obj.inlet.area);

            %evaluate gas volume fraction using correlation in BubblyFlow
            obj.gasVolFrac = hacsimulator.BubblyFlow.evalGasVolFrac(...
                superGasVel, superLiquidVel, obj.inlet.rhoGas, ...
                obj.inlet.rhoLiquid, obj.inlet.dia, obj.geometry.dirAngle);

        end %setGasVolumeFraction

        function setWallShear(obj)
            %SETWALLSHEAR Setter method for the wallShear property
            %   SETWALLSHEAR Determines the shear stress on the wall from
            %   the fluid flow and assigns the value to the property
            %   wallShear.
            %
            %CALL: obj.setWallShear()

            %{
                SY: Is this flexibility needed? Why not make the assumption
that
                wall friction applies only to water?
            %}

            if obj.controls.wallFriction == 0 %duct is frictionless
                obj.wallShear = 0;
            elseif obj.controls.wallFriction == 1 %wall friction applied to
both phases
                mDotTotal = obj.inlet.mDotGas + obj.inlet.mDotLiquid;

                %mass flowrate weighted average of the density of both
                %phases
                avgRho =
(obj.inlet.mDotLiquid/mDotTotal)*(obj.inlet.rhoLiquid + ...
```

```
                    obj.outlet.rhoLiquid)/2 +
(obj.inlet.mDotGas/mDotTotal)*...
                    (obj.inlet.rhoGas + obj.outlet.rhoGas)/2;

                %average velocity
                avgVelocity = mDotTotal/(avgRho*obj.outlet.area);

                %mass weighted average of the viscosity of both phases
                avgMu = (obj.inlet.mDotLiquid/mDotTotal)*(obj.inlet.muLiquid
+ ...
                    obj.outlet.muLiquid)/2 +
(obj.inlet.mDotGas/mDotTotal)*...
                    (obj.inlet.muGas + obj.outlet.muGas)/2;

                %average Reynolds number
                avgRe = avgRho*avgVelocity*obj.outlet.dia/avgMu;

                %determining friction factor
                fricFrac = obj.frictionFactor(avgRe);

                %setting wallShear
                obj.wallShear = (1/8)*fricFrac*avgRho*avgVelocity^2;

            else %otherwise wall friction loss applies to water only

                %arithmetic average of water density
                avgRho = 0.5*(obj.inlet.rhoLiquid + obj.outlet.rhoLiquid);

                %arithmetic average of water velocity
                avgVelocity = 0.5*(obj.inlet.liquidVel +
obj.outlet.liquidVel);

                %arithmetic average of water viscosity
                avgMu = 0.5*(obj.inlet.muLiquid + obj.outlet.muLiquid);

                %segment average Reynolds number
                avgRe = avgRho*avgVelocity*obj.outlet.dia/avgMu;

                %determining friction factor
                fricFrac = obj.frictionFactor(avgRe);

                %setting wallShear
                obj.wallShear = (1/8)*fricFrac*avgRho*avgVelocity^2;
            end
        end %setWallShear

        function dL = evalDiffusivity(obj, name)
            %EVALDIFFUSIVITY Determines the mass diffusivity of the given
species in water
            %   Diffusivities are evaluated based on conditions at the
            %   inlet using the Stokes-Einstein equation as recommended in
            %   Perry's Chemical Engineering Handbook
            %   (D*muLiquid/temp = constant).
```

```
            %
            %CALL: dL = obj.evalDiffusivity(name)
            %
            %INPUTS:
            %    (String) name - name of species to determine
            %       diffusivity
            %
            %OUTPUTS:
            %    (double) dL - diffusivity of speicies in water [m2/s]

            %get required parameters
            diffusivityFactor = obj.controls.diffusivityFactor;
            temp = obj.inlet.temp;
            muLiquid = obj.inlet.muLiquid;

            dL = hacsimulator.SpeciesData.evalMassDiffusivity(name, temp, ...
                muLiquid);

            dL = dL * diffusivityFactor;

        end %evalDiffusivity

        function fricFrac = frictionFactor(obj, numRe)
            %FRICTIONFACTOR evaluates the Colebrook equation to determine the
friction factor from a given Reynolds number
            %
            %CALL: fricFrac = obj.frictionFactor(numRe)
            %
            %INPUTS:
            %   (double) numRe - Reynolds number of the flow
            %
            %OUTPUTS:
            %   (double) fricFrac - friction factor

            %Determine the relative roughness
            relRoughness = obj.geometry.roughness/obj.outlet.dia;
            maxIter = 100;

            %Haaland equation to determine initial guess for friction factor
            fricFrac = (-1.8*log10(6.9/numRe + (relRoughness/3.7)^1.11))^(-
2);

            factorOK = false; %flag for tolerance check

            iter = 0;

            while factorOK == false
                %determine next approximation of f with Colebrook-White
equation
                nextF = (-2.0*log10(relRoughness/3.7 +
2.51/(numRe*sqrt(fricFrac))))^-2;

                %tolerance check
```

```
        if abs(nextF - fricFrac) <= 1e-6
            factorOK = true;
        end %tolerance check



        %update current approximation
        fricFrac = nextF;
        iter = iter +1;

        if iter > maxIter
            display(fricFrac);
            error('Max number of iterations reached')
        end

    end %while loop


end %frictionFactor

function nDotTransfer = evalMolTransfer(obj, name)
    %EVALMOLTRANSFER evaluates the the rate equation for mass
    %transfer from Chen & Rice (1983): nDotTransfer = K*C_LM*A
    %
    %CALL: nDotTransfer = obj.evalMolTransfer(name)
    %
    %INPUTS:
    %   (String) name - name of the species
    %
    %OUTPUTS:
    %   (double) nDotTransfer - rate of mol transfer of species
    %       given by K*C_LM*A

    %INTERFACIAL AREA:
    %A = 6 * gasVolFrac * segmentVolume /avgBubbleDia

    interfacialArea = (6 * obj.gasVolFrac / obj.avgBubbleDia)...
        * obj.geometry.volume;

    %MEAN CONCENTRATION DIFFERENCE

    %{
    % N.B. concDiffs should always be the same sign but concDiffOut
    % may switch signs from the Newton Raphson process. In that
    % case the arithmetic average is used because a log mean would
    % return a complex number.
    %}

    concDiffIn = obj.inlet.evalConcDiff(name);

    concDiffOut = obj.outlet.evalConcDiff(name);
```

```
            concDiffRatio = concDiffIn/concDiffOut;

            if concDiffRatio > 0
                %concDiffs have same sign, use log mean
                avgConcDiff = (concDiffIn - concDiffOut) / ...
                    log(concDiffRatio);

                %N.B. in MATLAB, log => ln

            else
                avgConcDiff = 0.5*(concDiffIn + concDiffOut);
            end

            %MASS TRANSFER COEFFICIENT

            %average exposure time
            exposureTimeIn = obj.inlet.evalBubbleDia() / ...
                abs(obj.inlet.slipVel);

            exposureTimeOut = obj.outlet.evalBubbleDia() / ...
                abs(obj.outlet.slipVel);

            avgExposureTime = 0.5*(exposureTimeIn + exposureTimeOut) ...
                * obj.controls.contactTimeFactor;

            %mass diffusivity
            diffusivity = obj.evalDiffusivity(name);

            %mass transfer coefficient given by Higbie (1935)
            massTransCoeff = 2*sqrt(diffusivity/(pi*avgExposureTime));

            %molar mass transfer rate, nDot = K * averageConcDiff * Area
            nDotTransfer = massTransCoeff * avgConcDiff * interfacialArea;

        end %evalMolTransfer

        function cD = evalDragCoefficient(obj)
            %EVALDRAGCOEFFICIENT Evaluates the drag coefficient for the
bubble swarm in the segment
            %   EVALDRAGCOEFFICIENT implements the approach from Rowe &
            %   Henwood (1961) outlined in Chen & Rice (1983) from current
            %   values held in the object's properties.
            %
            %CALL: cD = obj.evalDragCoefficient()
            %
            %OUTPUTS:
            %   (double) cD - drag coefficient for bubble swarm in
            %       downcomer segment


            %Determine particle Reynolds number
            partReNum = obj.evalParticleReNumber();
```

```
            cD = hacsimulator.BubblyFlow.evalDragCoeff(partReNum, ...
                obj.gasVolFrac);

        end %evalDragCoefficient

        function massResid = evalMassResid(obj)
            %EVALMASSRESID Evaluates the mass conservation residual from
current property values
            %
            %CALL: massResid = obj.evalMassResid()
            %
            %OUTPUTS:
            %   (double) massResid - the mass conservation residual

            %determine occupied areas of each phase at the outlet
            waterArea = obj.outlet.mDotLiquid / ...
                (obj.outlet.rhoLiquid * obj.outlet.liquidVel);

            gasArea = obj.outlet.area - waterArea;

            if obj.firstStage || ~(obj.controls.massTransfer ||
obj.controls.psychro)
                %in first stage, mass flowrate of gas phase is conserved
                massResid = obj.inlet.mDotGas - obj.outlet.rhoGas*...
                    (obj.outlet.liquidVel - obj.outlet.slipVel)*gasArea;
            else
                %in second stage, total mass flowrate is conserved and
                %interphase mass transfers are permitted
                massResid = obj.inlet.mDotGas + obj.inlet.mDotLiquid - ...
                    obj.outlet.rhoGas*(obj.outlet.liquidVel -
obj.outlet.slipVel)*gasArea...
                    - obj.outlet.rhoLiquid * obj.outlet.liquidVel *
waterArea;

            end %firstStage check

        end %evalMassResid

        function energyResid = evalEnergyResid(obj)
            %EVALENERGYRESID Evaluates the energy conservation residual from
current property values
            %    EVALENERGYRESID Evaluates the energy conservation residual
            %    with the sign convention that energy going into the control
            %    volume is positive and energy leaving the control volume is
            %    negative.
            %
            %CALL: energyResid = obj.evalEnergyResid()
            %
            %OUTPUTS:
            %    (double) energyResid - the energy conservation residual
```

```
            %get specific energies

            %inlet
            eWaterIn = obj.inlet.evalSpecificEnergies('liquid');
            eGasIn = obj.inlet.evalSpecificEnergies('gas');

            %outlet
            eWaterOut = obj.outlet.evalSpecificEnergies('liquid');
            eGasOut = obj.outlet.evalSpecificEnergies('gas');

            %evaluate energy residual
            energyResid = obj.inlet.mDotLiquid * sum(eWaterIn) + ...
                obj.inlet.mDotGas * sum(eGasIn) - ...
                obj.outlet.mDotLiquid * sum(eWaterOut) - ...
                obj.outlet.mDotGas * sum(eGasOut);

        end %evalEnergyResid

        function speciesResid = evalSpeciesResid(obj)
            %EVALSPECIESRESID Evaluates the conservation of species residuals
for each species under consideration
            %   evalSpeciesResid Evaluates the species conservation
            %   residual considering mass transfer from the gas phase to
            %   the liquid phase is positive since this is the expected
            %   behaviour.
            %
            %CALL: speciesResid = obj.evalSpeciesResid()
            %
            %OUTPUTS:
            %   (double[]) speciesResid - vector containing the species
            %       residual of each species ordered by the fluids property

            numComponents = length(obj.gases);
            molTransferRate = zeros(1, numComponents);

            for iComponent = 1:1:numComponents
                %evaluate mole transfer from mass transfer equation
                molTransferRate(iComponent) = ...
                    obj.evalMolTransfer(obj.gases{iComponent});
            end %calc loop

            %evaluate current numerical difference in molar flow rate
                deltaNDotLiquid = obj.outlet.mixture.getNDotLiquidArray()...
                    - obj.inlet.mixture.getNDotLiquidArray();


            speciesResid = molTransferRate - deltaNDotLiquid;

        end %evalSpeciesResid

        function momentumResid = evalMomentumResid(obj)
            %EVALMOMENTUMRESID Evaluates the conservation of momentum
residual from current property values
```

```
%   evalMomentumResid Evaluates the conservation of momentum
%   residual through a one-dimensional flow analysis where the
%   force and momentum vectors are evaulated relative to the
%   direction of the flow. A force vector or vector component in
the
%   same direction as the flow direction is considered
%   positive, otherwise it is considered negative while the
%   sign of the momenta are based relative to the outward
%   normals of the inlet and outlet control surfaces.
%
%CALL: momentumResid = obj.evalMomentumResid()
%
%OUTPUTS:
%   (double) momentumResid - the conservation of momentum
%       residual


%MOMENTA

%vectors assumed perpendicular to inlet and outlet surfaces

%inlet momenta are always negative (direction opposite of outward
%facing normal of inlet surface).

momWaterIn = -obj.inlet.mDotLiquid*obj.inlet.liquidVel;
momGasIn = -obj.inlet.mDotGas*(obj.inlet.liquidVel - ...
    obj.inlet.slipVel);

%outlet momenta are always positive (same direction as outward
facing
%normal of outlet surface)
momWaterOut = obj.outlet.mDotLiquid * obj.outlet.liquidVel;
momGasOut = obj.outlet.mDotGas*(obj.outlet.liquidVel - ...
    obj.outlet.slipVel);

sumOfMomenta = momWaterOut + momGasOut + momWaterIn + momGasIn;

%FORCES

%Fluid Wieghts

%gravitational acceleration, g [m/s2]
g = 9.80665;

%factor to multiply weights by to get components in dir of flow
weightFactor = -sin(obj.geometry.dirAngle);

%waterWeight = avgRho * g * waterVol * weightFactor
waterWeight = 0.5 * (obj.inlet.rhoLiquid + obj.outlet.rhoLiquid)
* ...
    g * obj.geometry.volume*(1 - obj.gasVolFrac) * weightFactor;

%gasWeight = avgRho * g * gasVol * weightFactor
```

```
            gasWeight = 0.5 * (obj.inlet.rhoGas + obj.outlet.rhoGas) *...
                g * obj.geometry.volume * obj.gasVolFrac * weightFactor;

            %Pressure Forces

            %assumption: pressures of both fluids are equal at each point in
downcomer

            %pressure force at inlet always with flow
            presForceIn = obj.inlet.pressure*obj.inlet.area;

            %pressure force at outlet always resists flow
            presForceOut = -obj.outlet.pressure*obj.outlet.area;

            %factor to multiply wall pressure force by to get component in
dir of flow
            wallPresFactor = cos(obj.geometry.wallAngle);

            %wallPresFactor = 0 when inlet and outlet diameters are equal

            if obj.inlet.dia > obj.outlet.dia %when duct is convergent,
preForceWall resists flow
                wallPresFactor = -wallPresFactor;
            end %convergent duct check

            %presForceWall = avgPressure * wallArea * wallPresFactor
            presForceWall = 0.5*(obj.inlet.pressure +
obj.outlet.pressure)*...
                obj.geometry.wallArea*wallPresFactor;

            %Friction Force

            %Set wall shar stress from updated properties
            obj.setWallShear()

            %factor to multiply friction force by to get component in dir of
flow
            fricForceFactor = -sin(obj.geometry.wallAngle); %-ve b/c wall
friction always resists flow

            %fricForceFactor is -ve b/c wall friction always resists flow

            %frictionForceWall = wallShear * wallArea * fricForceFactor
            fricForceWall = obj.wallShear*obj.geometry.wallArea*...
                fricForceFactor;

            sumOfForces = waterWeight + gasWeight + presForceIn +
presForceOut + ...
                presForceWall + fricForceWall;

            momentumResid = sumOfMomenta - sumOfForces;
```

```
        end %evalMomentumResid

        function slipVelResid = evalSlipVelocityResid(obj)
            %EVALSLIPVELOCITYRESID Evaluates the slip velocity residual based
current values of properties
            %   EVALSLIPVELOCITYRESID Evaluates the slip velocity residual
            %   based upon the balance of buoyancy and drag forces on a
            %   bubble (Source: Chen & Rice 1983)
            %
            %CALL: slipVelResid = obj.evalSlipVelocityResid()
            %
            %OUTPUTS:
            %   (double) slipVelResid - residual of slip velocity equation

            %{
            % N.B. Slip velocity sign convention:
            % The sign convention for the slip velocity is flow direction
            % positive. If the relative velocity is in the same direction
            % as the flow with a stationary frame of reference the slip
            % velocity is positive, otherwise it is negative.
            %}


            %gravitational acceleration, g [m/s2]
            g = 9.80665;

            %bubble drag cofficient
            dragCoeff = obj.evalDragCoefficient();

            slipVelResid = obj.outlet.slipVel^2 -
(4/3)*((obj.outlet.rhoLiquid ...
                - obj.outlet.rhoGas)/obj.outlet.rhoLiquid)* (g/dragCoeff)*...
                obj.avgBubbleDia;

        end %evalSlipVelocityResid

        function particleReNum = evalParticleReNumber(obj)
            %EVALPARTICLERENUM Evaluates the particle Reynolds number at the
segment outlet
            %
            %CALL: particleReNum = obj.evalParticleReNumber()
            %
            %OUTPUTS:
            %   (double) particleReNum - particle Reynolds number based
            %       upon the fluid properties at the outlet.

            particleReNum = (obj.outlet.rhoLiquid - obj.outlet.rhoGas) *...
                abs(obj.outlet.slipVel) *
obj.avgBubbleDia/obj.outlet.muLiquid;

        end %evalParticleReNum

    end %private methods
```

```
end %DowncomerSegment
```

A.5 DowncomerSection

```
classdef DowncomerSection < hacsimulator.MasterHelp
    %DOWNCOMERSECTION This object contains the relevant properties and
methods for a section of the downcomer
    %   DowncomerSection stores the solution variables (pressure,
    %   temperature, water velocity, slip velocity, and gas composition),
    %   fluid properties (density and viscosity), diameter, elevation, area
    %   and mass flowrates.
    %
    %   DowncomerSection requires that the refprop and CoolProp MATLAB
    %   scripts be located either in the current directory or a subfolder
    %   of the current directory.
    %
    %Author: Stephen Young, MIRARCO, Sudbury (syoung@mirarco.org)
    %Date: April 2016

    %{
        N.B. Property characters and fluid names for refprop an CoolProp

        Property legend (refprop and CoolProp)
            D - Density
            V - Viscosity
            U - Specific internal energy

        Fluid names - The following fluid names work for when looking up
        properties in either refprop or CoolProp (Chemical formula => name):
            N2 => nitrogen
            O2 => oxygen
            Ar => argon
            CO2 => CO2
            H2O => water

    %}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    properties (SetAccess = private)
        %Solution variables

        pressure; %double - pressure of the gas and water [Pa]
        temp; %double - temperature of the gas and water [K]
        liquidVel; %double - velocity of water [m/s]
        slipVel; %double - relative velocity of the gas [m/s]
        mixture; %SpeciesTracker - SpeciesTracker object for tracking molar
flowrates and gas composition

        %Fluid properties

        rhoLiquid; %double - density of water [kg/m3]
```

```
        muLiquid; %double - viscosity of water [Pa-s]
        rhoGas; %double - density of gas mixture [kg/m3]
        muGas; %double - viscosity of gas mixture [Pa-s]

        %geometry

        dia; %double - inside diameter of downcomer at current section [m]
        zed; %double - elevation of the current section [mAD]
        area; %double - flow area of the downcomer at current section [m2]

        %flow parameters

        mDotLiquid; %double - mass flowrate of liquid phase at section [kg/s]
        mDotGas; %double - mass flowrate of gas phase at section [kg/s]
        bubbleFlux; %double - bubble flux at given section [bubbles/s]

    end %gettable properties

    properties (Access = protected, Constant)

        %char[] property characters for refprop & CoolProp
        PROPS = ['D', 'V'];



        %{
        % Having SOLUTION_VARS property replaces the getSolnVars method and
        % saves computational effort.
        %}

        %String{} - names of hydrodynamic solution variables
        SOLUTION_VARS = {'pressure', 'temp', 'liquidVel', 'slipVel'};

        %{
        % Having the LIQUID constant private property is a bit of a cheat..
        % Need to invesitgate a good way to make this more flexible. Is is
        % as simple as adding an argument to be passed from Downcomer to
        % DowncomerSegment to this?
        %}

    end %private constant properties

    properties (Access = protected)

        liquidComponents; %LiquidPhase - object describing components of the
liquid phase

        %controls
        massTransfer; %mass transfer included if TRUE, otherwise excluded
        psychro; %psychrometry included if TRUE, otherwise excluded
        refpropFlag; %use refprop if TRUE, otherwise use CoolProp
```

```
        firstStage = true; %boolean - in first stage if true, otherwise
second stage of solution

    end

    methods %PUBLIC METHODS

        %CONSTRUCTOR
        function obj = DowncomerSection(diameter, flowArea, elevation,
state,...
                mDotLiquid, liquidComponents, mDotGas, gasMixture, ...
                bubbleFlux, controls)
            %Constructs object, sets the values of all properties from given
values
            %
            %CALL: obj = DowncomerSection(diameter, flowArea, elevation, ...
            %       state, mDotLiquid, liquidComponents, mDotGas,
gasMixture,...
            %       bubbleFlux, controls)
            %
            %INPUTS:
            %   (double) diameter - diameter of downcomer duct at given
            %       section [m]
            %   (double) flowArea - area available for flow at section [m2]
            %   (double) elevation - elevation of given section [mAD]
            %   (double[]) state - vector with pressure [Pa], temperature
            %       [K], veloctiy of liquid phase [m/s], and slip veloctiy
            %       of gas phase [m/s]
            %   (double) mDotLiquid - mass flow rate of the liquid phase
            %       [kg/s]
            %   (LiquidPhase) liquidComponents - LiquidPhase object with
            %       names of bulk fluid and cosolute in liquid phase and
            %       cosolute concentration
            %   (double) mDotGas - mass flow rate of the gas phase [kg/s]
            %   (SpeciesTracker) gasMixture - SpeciesTracker object for
            %       this section
            %   (double) bubbleFlux - bubble flux accross the section
            %       [bubbles/s]
            %   (ControlParameters) controls - handle of controls property
            %      from Downcomer

            %{
            % addpath('.\Refprop','.\CoolProp');
            % Refprop and CoolProp calls are now handled by the
            % PropertyCalculator class. Package structure eliminates need
            % for the addpath call.
            %}

            %get control parameters
            obj.massTransfer = controls.massTransfer;
            obj.psychro = controls.psychro;
            obj.refpropFlag = controls.refpropFlag;
```

```
            obj.setState(state);
            obj.mixture = gasMixture.deepCopy();
            obj.liquidComponents = liquidComponents;


            %Determine and set fluid properties
            obj.setSectionProperties()

            %Set geometric parameters
            obj.dia = diameter;
            obj.area = flowArea;
            obj.zed = elevation;

            %Set mass flowrates
            obj.setMassFlows(mDotLiquid, mDotGas)
            obj.bubbleFlux = bubbleFlux;

        end %CONSTRUCTOR

        function updateState(obj, newState)
            %UPDATESTATE Setter method to update the solution variables at
the section
            %
            %CALL: obj.updateState(newState, firstStage)
            %
            %INPUTS:    (double[]) newState - vector with the solution
variables
            %           (boolean) firstStage - mol flowrates constant if
true, otherwise
            %               update mol flowrates.

            obj.setState(newState)

            if ~obj.firstStage && (obj.massTransfer || obj.psychro)
                if obj.massTransfer
                    %massTransfer = true => newState includes molar flowrates
                    obj.mixture.updateComposition(newState(length(...
                        obj.SOLUTION_VARS)+1:end), obj.pressure, obj.temp);
                else
                    %{
                        massTransfer = false, psychro = true => newState does
                        not include molar flowrates but absHumidity and
                        composition properties need to be updated based on
new
                        pressure and temperature.

                        SpeciesTraker/updateComposition is called using its
                        own current values of nDotGas.
                    %}
                    nDotGas = obj.mixture.getNDotGasArray;

                   obj.mixture.updateComposition(nDotGas, obj.pressure, ...
                       obj.temp);
```

```
            end %massTransfer check

        end %firstStage/massTransfer/psychro check

    end

    function updateFluidProps(obj)
        %UPDATEFLUIDPROPS Update fluid properties based upon current
stored values of pressure, temperature, and gas composition
        %
        %CALL: obj.updateFluidProps()

        obj.setSectionProperties();

    end %updateFluidProps

    function diameter = evalBubbleDia(obj)
        %EVALBUBBLEDIA Evaluates bubble diameter from current values of
bubble mass and gas density
        %
        %CALL: diameter = obj.evalBubbleDia()
        %
        %OUPUTS:
        %   (double) diameter - average bubble diameter [m]


        bubbleMass = obj.mDotGas / obj.bubbleFlux;

        volume = bubbleMass/obj.rhoGas;

        diameter = (6*volume/pi)^(1/3);

    end %getBubbleDia

    function concDiff = evalConcDiff(obj, name)
        %EVALCONCDIFF Evaluates the concentration difference of a given
species at the section
        %
        %CALL: concDiff = obj.evalConcDiff(name)
        %
        %INPUTS:
        %   (string) name - name of species
        %
        %OUTPUTS:
        %   (double) concDiff - concentration difference of species
        %       at section [mol/m3].

        bulkConc = obj.evalBulkConc(name);

        %INTERFACIAL CONCENTRATION
        henryConstant = hacsimulator.SpeciesData.evalHenrysConstant(...
```

```
                name, obj.temp, obj.liquidComponents);

            molFrac = obj.mixture.getMolFrac(name, obj.psychro);

            interConc = henryConstant * obj.pressure * molFrac;

            %CONCENTRATION DIFFERENCE
            concDiff = interConc - bulkConc;

        end %evalConcDiff

        function bulkConc = evalBulkConc(obj, name)
        %EVALBULKCONC Evaluates the bulk concentration of a given gas solute
at the section
        %
        %CALL: bulkConc = obj.evalBulkConc(name)
        %
        %INPUTS:
        %   (String) name - name of species
        %
        %OUTPUTS:
        %   (double) bulkConc - bulk concentration of the gas solute in the
        %       liquid phase [m3]

            nDotLiquid = obj.mixture.getNDotLiquid(name);

            bulkConc = obj.rhoLiquid * nDotLiquid / obj.mDotLiquid;

        end %evalBulkConc

        function updateMassFlows(obj, newMDotLiquid, newMDotGas)
           %UPDATEMASSFLOWS Setter method for the mass flowrates of each
phase
           %
           %CALL: obj.updateMassFlows(newMDotLiquid, newMDotGas)
           %
           %INPUTS:
           %    (double) newMDotLiquid - mass flowrate of water to assign to
           %        mDotLiquid property
           %    (double) newMDotGas - mass flowrate of gas to assign to
           %        mDotGas property
           %
           %THROWS: illegalArgument exception if the inputs to the setter
           %    method would change the total mass flowrate through the
           %    system more than an acceptable tolerance of 1e-10.

           %display(num2str(obj.mDotLiquid + obj.mDotGas,8),
'currentTotalMassFlow')
           %display(num2str(newMDotLiquid + newMDotGas,8),
'newTotalMassFlow')
           if abs((newMDotLiquid + newMDotGas) - ...
                   (obj.mDotLiquid + obj.mDotGas)) > 1e-10
```

```
                error('DowncomerSection:setMassFlows:illegalArgument',...
                    'Something went wrong! You are creating or destroying
mass!')

            end %input sanity check

            obj.setMassFlows(newMDotLiquid, newMDotGas)

        end

        function state = getStateVector(obj)
            %GETSTATEVECTOR Getter method for the state variables at the
downcomer section
            %
            %CALL: state = obj.getStateVector(firstStage)
            %
            %INPUTS:
            %   (boolean) firstStage - exclude mol flowrates if true,
            %       otherwise, include
            %
            %OUTPUTS:
            %   (double[]) state - vector containing the solution
            %       variables

            numSolutionVars = length(obj.SOLUTION_VARS);
            state = zeros(1,numSolutionVars);

            for iSolutionVar = 1:1:numSolutionVars
               state(iSolutionVar) = obj.(obj.SOLUTION_VARS{iSolutionVar});
            end

            if ~obj.firstStage && obj.massTransfer
                %append molar flowrates in gas phase when massTransfer is
                %true
                state = [state, obj.mixture.getNDotGasArray()];
            end %massTransfer check

        end %getStateVector

        function specificEnergies = evalSpecificEnergies(obj, phase)
            %EVALSPECIFICENERGIES Evaluates the specific internal, kinetic,
potential and pressure energies of the given phase
            %
            %CALL: specificEnergies = obj.evalSpecificEnergies(phase)
            %
            %INPUTS:
            %   (String) phase - the phase (liquid or gas) to eval specific
            %       energies
            %
            %OUTPUTS:
            %   (double[]) specificEnergies - vector containing the
            %       specific internal, kinetic, potential and pressure
            %       energies of given phase
            %
```

```
            %THROWS:
            %   (ME) illegalArgument - if phase is not 'liquid' or 'gas'

            %input checking
            if ~(strcmp(phase, 'liquid') || strcmp(phase,'gas'))

error('DowncomerSection:evalSpecificEnergies:illegalArgument',...
                    'The phase ''%s'' is not recognised.', phase)
            end %phase check

            %gravitational acceleration, g [m/s2]
            g = 9.80665;

            internalEnergy = obj.evalInternalEnergy(phase);
            potentialEnergy = g * obj.zed;

            if strcmp(phase, 'liquid')

                kineticEnergy = 0.5*obj.liquidVel^2;
                pressureEnergy = obj.pressure / obj.rhoLiquid;

            else %phase = 'gas'

                kineticEnergy = 0.5 * (obj.liquidVel - obj.slipVel)^2;
                pressureEnergy = obj.pressure / obj.rhoGas;

            end %phase check

            specificEnergies = [internalEnergy, kineticEnergy, ...
                potentialEnergy, pressureEnergy];

        end %evalSpecificEnergies

        function secondStage(obj)
           obj.firstStage = false;
        end %secondStage

    end %public methods

    methods (Access = protected)
        %methods used internally by the SectionProperties object

        function setState(obj, state)
            %SETSTATE Setter method to set the solution variables
            %   SETSTATE Sets the pressure, temp, liquidVel, slipVel
            %   properties
            %
            %CALL: obj.setState(state)
            %
            %INPUTS:
            %   (double[]) state - vector containing the pressure [Pa],
            %       temperature [K], water velocity [m/s], gas slip
            %       velocity [m/s]
```

```
            for iSolutionVar = 1:1:length(obj.SOLUTION_VARS)
                obj.(obj.SOLUTION_VARS{iSolutionVar}) = state(iSolutionVar);
            end

        end %setState

        function setMassFlows(obj, mDotLiquid, mDotGas)
           %SETMASSFLOWS Setter method for the mass flowrates of each phase
           %
           %CALL: obj.setMassFlows(mDotLiquid, mDotGas)
           %
           %INPUTS:
           %    (double) mDotLiquid - mass flowrate of water to assign to
           %        mDotLiquid property
           %    (double) mDotGas - mass flowrate of gas to assign to
           %        mDotGas property
           %
           %THROWS: illegalArgument exception if either inputs are not real
           %    scalar numbers.


           %display(num2str(obj.mDotLiquid + obj.mDotGas,8),
'currentTotalMassFlow')
           %display(num2str(newMDotLiquid + newMDotGas,8),
'newTotalMassFlow')


           %Input sanity check
           if ~hacsimulator.NumberCheck.isNumber(mDotLiquid)
               error('DowncomerSection:setMDotLiquid:illegalArgument',...
                   'mDotLiquid must be a real scalar number');
           elseif ~hacsimulator.NumberCheck.isNumber(mDotGas)
               error('DowncomerSection:setMDotGas:illegalArgument',...
                   'mDotGas must be a real scalar number');

           end %input check



           obj.mDotLiquid = mDotLiquid;
           obj.mDotGas = mDotGas;

        end %setMassFlows

        function setSectionProperties(obj)
            %SETSECTIONPROPERTIES Setter method for the fluid properties
            %   SETSECTIONPROPERTIES Determines and sets the values of
            %   density and viscosity of each phase from the current
            %   pressure [Pa], temperature [K] and gas composition
            %
            %CALL: obj.setSectionProperties()
```

```
            %evaluate properties
            liquidProp = obj.evalLiquidProps(obj.PROPS);
            gasProp = obj.evalGasProps(obj.PROPS);

            %set object properties
            obj.rhoLiquid = liquidProp(1);
            obj.muLiquid = liquidProp(2);

            obj.rhoGas = gasProp(1);
            obj.muGas = gasProp(2);

        end %setSectionProperties

        %{
            evalLiquidProps and evalMixtureProps use the argument PROPS so
            they have more flexibility
        %}

        function liquidProps = evalLiquidProps(obj, props)
            %EVALLIQUIDPROPS Evaluates the desired properties of the liquid
phase using refprop or CoolProp functions.
            %   EVALLIQUIDPROPS is used by setSectionProperties and
            %   evalInternalEnergy to determine relevant properties of the
            %   liquid phase.
            %
            %CALL: liquidProps = obj.evalLiquidProps(props)
            %
            %INPUTS:
            %   (char[]) props - a char vector with the property characters
            %      for refprop and CoolProp functions
            %
            %OUTPUTS:
            %   (PropertyCalculator) liquidProps - a PropertyCalculator
            %       object with desired properties

            liquidName = {obj.liquidComponents.bulkFluid};

            liquidProps =
hacsimulator.PropertyCalculator.evalPureFluidProp(...
                props, obj.pressure, obj.temp, liquidName, obj.refpropFlag);

        end %evalLiquidProps function

        function gasProps = evalGasProps(obj, props)
            %EVALMIXTUREPROPS Evaluates the desired properties of the gas
phase using refprop or CoolProp functions.
            %   EVALMIXTUREPROPS is used by setSectionProperties and
            %   evalInternalEnergy to determine relevant properties of the
            %   gas phase.
            %
            %CALL: mixtureProps = obj.evalMixtureProps(props)
            %
            %INPUTS:
            %   (char[]) props - a char vector with the property characters
```

```
            %      for refprop and CoolProp functions
            %
            %OUTPUTS:
            %   (PropertyCalculator) gasProps - a PropertyCalculator object
            %       with desired properties



            if obj.firstStage

                speciesNames = obj.mixture.getNameArray(false);

                if obj.refpropFlag
                   mixtureFractions = obj.mixture.getMassFracArray(false);

                else
                    mixtureFractions = obj.mixture.getMolFracArray(false);
                end

            else

                speciesNames = obj.mixture.getNameArray(obj.psychro);
                if obj.refpropFlag
                   mixtureFractions =
obj.mixture.getMassFracArray(obj.psychro);

                else
                    mixtureFractions =
obj.mixture.getMolFracArray(obj.psychro);
                end

            end

            gasProps = hacsimulator.PropertyCalculator.evalMixtureProp(...
                props, obj.pressure, obj.temp, speciesNames, ...
                mixtureFractions, obj.refpropFlag);



        end %evalMixtureProps

        function internalEnergy = evalInternalEnergy(obj, phase)
           %EVALINTERNALENERGY evaluates the value of internal energy the
           %givenphase at the current state
           %
           %CALL: internalEnergy = obj.evalInternalEnergy(phase)
           %
           %INPUTS:
           %    (string) phase - the phase (liquid or gas) to determine
           %        the internal energy
           %
           %OUTPUTS:
```

```
            %     (double) internalEnergy - the internal energy of the
            %         given phase [J/kg]
            %
            %THROWS:
            %     (MException) illegalArgument - illegalArgument if phase is
            %         not 'gas' or 'liquid'

            CHAR_ID = 'U';

            if strcmp(phase, 'liquid')
                internalEnergy = obj.evalLiquidProps(CHAR_ID);
            elseif strcmp(phase, 'gas')
                internalEnergy = obj.evalGasProps(CHAR_ID);
            else

error('DowncomerSection:evalInternalEnergy:illegalArgument',...
                    'The phase ''%s'' is not recognised.', phase)
            end %phase check

        end %evalInternalEnergy

    end %private methods

end %DowncomerSection
```

A.6 SpeciesTracker

```
classdef SpeciesTracker < hacsimulator.MasterHelp
    %SPECIESTRACKER Tracks the composition, and necessary parameters of the
gas
    %mixture in the downcomer absorption process.
    %   SpeciesTracker tracks the molar flowrates of each species under
    %   consideration, mol fraction of each species in the gaseous phase,
    %   molar mass of the mixture. SpeciesTracker also stores the values of
    %   molar mass of each species under consideration. Currently only able
    %   to deal with the common species in air: N2, O2, Ar, CO2 and H2O.
    %
    %Author: Stephen Young, MIRARCO, Sudbury, ON (syoung@mirarco.org)
    %Date: April 2016

    properties (SetAccess = protected)

        %GasSpecies[] - array of GasSpecies objects for each dry component
        components = hacsimulator.GasSpecies.empty;
        numComponents; %int - the number of species in the mixture

        %mixture properties

        dryMolMass; %double - molar mass of dry gas mixture [kg/mol]
        humidMolMass; %double - molar mass of the gas mixture [kg/mol]

        absHumidity; %double - absolute humidity of the gas mixture [kg H2O /
kg dry air]
```

```
    end %protected properties

    properties (Constant)

        %double - relative humidity of the gas mixture [Pa/Pa]
        RELATIVE_HUMIDITY = 1;

        %a relative humidity of 100% is assumed in downcomer and riser

    end %constant property

    properties (Access = protected)
        refpropFlag; %boolean - use refprop if TRUE, otherwise use CoolProp
        psychro; %boolean - include psychrometric calculations if TRUE,
otherwise exclude
        nameArray; %String{}
        molarAbsHumidity; %double - absolute humidity in [mol H2O / mol dry
air]
    end

    methods
        %CONSTRUCTOR
        function obj = SpeciesTracker(gasMixture, pressure, temp, controls)
            %SPECIESTRACKER Constructs a SpeciesTracker object, assigning
values to all properties
            %
            %CALL: obj = SpeciesTracker(gasMixture, pressure, temp, controls)
            %
            %INPUTS:
            %   (GasSpecies) gasMixture -
            %   (double) pressure - pressure at current section [Pa]
            %   (double) temp - temperature at current section [K]
            %   (ControlParameters) controls - ControlParameters object
            %       from DowncomerSection

            %{
            % addpath('.\Refprop', '.\CoolProp')
            % All refprop and CoolProp calls are now done through the
            % PropertyCalculator class. No longer need to add the
            % Refprop and CoolProp directories to the path in this class
            %}

            switch nargin

                case 1 %components is a SpeciesTracker object for deep copy

                    if ~isa(gasMixture, 'hacsimulator.SpeciesTracker')
                        error('SpeciesTracker:Constructor:illegalArgument',
...
                            'components must be of type SpeciesTracker')
                    end %input check
```

```
                obj.numComponents = gasMixture.numComponents;

                for iComponent = 1:1:gasMixture.numComponents

                    obj.components(iComponent) = ...
                        gasMixture.components(iComponent).deepCopy();

                end

                obj.dryMolMass = gasMixture.dryMolMass;
                obj.humidMolMass = gasMixture.humidMolMass;

                obj.absHumidity = gasMixture.absHumidity;


                obj.molarAbsHumidity = gasMixture.molarAbsHumidity;

                %get control parameters
                obj.refpropFlag = gasMixture.refpropFlag;
                obj.psychro = gasMixture.psychro;


            case 4 %components is a GasSpecies[] for normal construction

                %input sanity check
                if ~isa(gasMixture, 'hacsimulator.GasSpecies')
                    error('SpeciesTracker:Constructor:illegalArgument',
...
                        'components must be of type GasSpecies')
                elseif sum([gasMixture.dryMolFrac]) ~= 1
                    error('SpeciesTracker:Constructor:illegalArgument',
...
                        'mole fractions must sum to 1')
                elseif ~hacsimulator.NumberCheck.isNumber(pressure)
                    error('SpeciesTracker:Constructor:illegalArgument',
...
                        'pressure must be a real scalar number')
                elseif ~hacsimulator.NumberCheck.isNumber(temp)
                    error('SpeciesTracker:Constructor:illegalArgument',
...
                        'temperature must be a real scalar number')
                elseif ~isa(controls, 'hacsimulator.ControlParameters')
|| ~isscalar(controls)
                    error('SpeciesTracker:Constructor:illegalArgument',
...
                        'controls must be of type ControlParameters')
                end



                %Determine number of species under consideration
```

```
                    obj.numComponents = length(gasMixture);


                    %initialise species array
                    for iComponent = 1:1:obj.numComponents

                        obj.components(iComponent) =
gasMixture(iComponent).deepCopy();

                    end


                    %get necessary control parameters
                    obj.refpropFlag = controls.refpropFlag;
                    obj.psychro = controls.psychro;

                    obj.setComposition(pressure, temp);

            end

            obj.nameArray = {obj.components.name};

        end %constructor

        function updateComposition(obj, nDotGas, pressure, temp)
            %UPDATECOMPOSITION Method for updating the molar flowrates of a
species in both
            %phases as part of the Newton-Raphson numerical procedure
            %
            %CALL: obj.updateComposition(nDotGas, pressure, temp, refprop)
            %
            %INPUTS:
            %   (double[]) nDotGas - molar flowrates of species in gaseous
            %       phase in same ordered by species fluid string
            %   (double) pressure - pressure of gas mixture [Pa]
            %   (double) temp - temperature of gas mixture [K]
            %
            %THROWS: error if nDotGas does not have the same number of
            %   elements as the species cell array.

            %Input sanity check
            if length(nDotGas) > obj.numComponents
                error('SpeciesTracker:updateMolFlows:illegalArgument',...
                    'A wild species has appeared! Too many molar flowrates')
            elseif length(nDotGas) < obj.numComponents
                error('SpeciesTracker:updateMolFlows:illegalArgument',...
                    'A species has disappeared! Too few molar flowrates')
            end %input check

            for iComponent = 1:1:obj.numComponents


obj.components(iComponent).updateMolFlows(nDotGas(iComponent))
```

```
            newDryMolFrac = nDotGas(iComponent) / sum(nDotGas);

            obj.components(iComponent).setDryMolFrac(newDryMolFrac);

        end %for loop

        obj.setComposition(pressure, temp);

    end %updatComposition

    function speciesTrackerCopy = deepCopy(obj)

        speciesTrackerCopy = hacsimulator.SpeciesTracker(obj);

        speciesTrackerCopy.setHumidMolFracs();

    end %deepCopy

    %accessor methods

    function molFrac = getMolFrac(obj, name, humid)
        %GETMOLFRAC Getter method for the mole fraction of a given
species
        %
        %CALL:  molFrac = obj.getMolFrac(name, humid)
        %
        %INPUTS:
        %   (String) name - name of the species
        %   (boolean) humid - get humid mole fraction if true,
        %       otherwise get dry mole
        %
        %OUTPUTS:
        %   (double) molFrac - mole fraction of given species

        index = obj.getComponentIndex(name);

        if humid
            molFrac = obj.components(index).humidMolFrac;
        else
            molFrac = obj.components(index).dryMolFrac;
        end %humid check


    end %getMolFrac

    function molMass = getMolMass(obj, name)
        %GETMOLMASS Gets molar mass of a given species
        %
        %CALL: molMass = obj.getMolMass(name)
        %
        %INPUTS:
```

```
            %   (String) name - name of species to retrieve molar mass
            %
            %OUTPUTS:
            %   (double) molMass - molar mass of given species

            index = obj.getComponentIndex(name);

            molMass = obj.components(index).molMass;

        end %getMolMass

        function nDotLiquid = getNDotLiquid(obj, name)
            %GETNDOTLIQUID Getter method for the molar flowrate of the given
species in water
            %
            %CALL: nDotLiquid = obj.getNDotLiquid(name)
            %
            %INPUTS:
            %    (string) name - name of desired species
            %
            %OUTPUTS:
            %    (double) nDotWater - molar flowrate of species in liquid
            %        phase [mol/s]

            nDotLiquid =
obj.components(obj.getComponentIndex(name)).nDotLiquid;

        end %getNDotWater

        function waterMassFrac = getWaterMassFrac(obj)
            %GETWATERMASSFRAC Getter method for the mass fraction of water
            %
            %CALL: waterMassFrac = obj.getWaterMassFrac()
            %
            %OUTPUTS:
            %   (double) waterMassFrac- current mass fraction of water in
            %       humid gas mixture

            waterMassFrac = obj.absHumidity / (1 + obj.absHumidity);

        end %getWaterMassFrac

        function nDotGasArray = getNDotGasArray(obj)
            %GETNDOTGASARRAY Getter method for an array of the values of
nDotGas in the
            %GasSpecies array
            %
            %CALL: nDotGasArray = obj.getNDotGasArray()
            %
            %OUTPUTS:
            %   (double[]) nDotGasArray - array containing molar flowrates
            %       of each dry species in the gas phase
```

```
            %use MATLAB array format to get nDotGas array
            nDotGasArray = [obj.components.nDotGas];

        end %getNDotGasArray

        function molFracArray = getMolFracArray(obj, humid)
            %GETMOLFRAC is a getter method for an array containing the mol
fraction of each species
            %
            %CALL: molFracs = obj.getMolFracArray(humid)
            %
            %INPUTS:
            %   (boolean) humid - get humid mol fracs if true, dry if false
            %
            %OUTPUTS:
            %   (double[]) molFracs - array with the mol fractions of each
            %       species in the mixture

            if humid

                molFracArray = [obj.components.humidMolFrac];

               %evaluate mole fraction of water vapour
               molFracArray(end+1) = 1 - sum(molFracArray);

            else

                molFracArray = [obj.components.dryMolFrac];

            end %humid check

        end %getMolFracArray

        function massFracArray = getMassFracArray(obj, humid)
            %GETMASSFRACARRAY Evaluates the mass fraction of each species in
the gas mixture
            %and returns it in an array ordered by the species fluid string.
            %
            %CALL: massFracs = obj.getMassFracArray(humid)
            %
            %INPUTS:
            %    (boolean) humid - return humidMassFracs if true,
            %            otherwise return dryMassFracs
            %
            %OUTPUTS:
            %    (double[]) massFracs - array containing the mass fraction
            %        of each species

            %initialise massFracs array
            molFracArray = obj.getMolFracArray(humid);
            molarMassArray = obj.getMolMassArray();
```

```
            if humid

                MOLAR_MASS_WATER = 18e-3;

                molarMassArray(end+1) = MOLAR_MASS_WATER;

                %calculate humid mass fracs of dry components
                massFracArray = molFracArray .* molarMassArray / ...
                    obj.humidMolMass;

            else

                massFracArray = molFracArray .* molarMassArray / ...
                    obj.dryMolMass;

            end %humid check

        end %getMassFracArray

        function molMassArray = getMolMassArray(obj)
            %GETMOLMASSARRAY Getter method for the molar mass of each species
            %
            %CALL: molMasses = obj.getMolMassArray()
            %
            %OUTPUTS:
            %   (double[]) molMasses - array containing the molar mass of
            %       each species

            %initialise molar mass array
            molMassArray = [obj.components.molMass];

        end %getMolMassArray

        function nDotLiquidArray = getNDotLiquidArray(obj)
            %GETNDOTLIQUIDARRAY Getter method for an array of the values of
nDotLiquid in the GasSpecies array
            %
            %CALL: nDotLiquidArray = obj.getNDotLiquidArray()
            %
            %OUTPUTS:
            %   (double[]) nDotLiquidArray - array containing the values of
            %       nDotLiquid for each GasSpecies object in components

            %initialise nDotGas array
            nDotLiquidArray = [obj.components.nDotLiquid];

        end %getNDotLiquidArray

        function nameArray = getNameArray(obj, humid)
            %GETNAMEARRAY Getter method for an array of the values of name in
the GasSpecies array
            %
            %CALL: nameArray = obj.getNameArray()
```

```
            %
            %INPUTS:
            %   (boolean) humid - include water in mixture if true,
            %       otherwise exclude
            %
            %OUTPUTS:
            %   (String{}) nameArray - cell array containing the values of
            %       name for each GasSpecies object in components

            %initialise nDotGas array
            nameArray = obj.nameArray;


            if humid
               nameArray(end+1) = {'water'};
            end

        end %getNameArray

    end %public methods

    methods (Access = protected)

        function index = getComponentIndex(obj, search)
            %GETCOMPONENTINDEX Method to determine the index of a given
species in the GasSpecies array
            %
            %CALL: index = obj.getComponentIndex(search)
            %
            %INPUTS:
            %   (string) search - string with name of speices to search for
            %
            %THROWS:
            %   (MException) noSuchElement - noSuchElement if search is not
            %       found in components

            index = find(strcmp(obj.nameArray, search));

            if isempty(index)
                error('SpeciesTracker:getComponentIndex:noSuchElement',...
                    '%s is not in the mixture', search)
            end
        end

        %setter methods

        function setComposition(obj, pressure, temp)
            %SETCOMPOSITION Setter method for the composition properties
            %   SETCOMPOSITION Method to set all composition properties to
            %   to be used with SpeciesTracker constructor and updateState
            %   methods.
            %
            %CALL: obj.setComposition(pressure, temp, refprop)
```

```
            %
            %INPUTS:
            %   (double) pressure - pressure of the mixture [Pa]
            %   (double) temp - temperature of the mixture [K]
            %   (boolean) refprop - use refprop if true, otherwise use
            %       CoolProp

            %Determine molarAbsHumidity of mixture
            obj.setMolarAbsHumidity(pressure, temp);

            %Set humid mole fractions
            obj.setHumidMolFracs();

            %Set dry mixture molar masss
            obj.setDryMolMass();

            %Set humid molar mass
            obj.setHumidMolMass();

            %Set absolute humidity [kg H2O / kg dry air]
            obj.setAbsHumidity();


        end %setComposition

        function setMolarAbsHumidity(obj, pressure, temp)
            %SETMOLARABSHUMIDTY Setter method for the molarAbsHumidity
property
            %   setMolarAbsHumidity Determines and sets the value of
            %   molarAbsHumidity based upon the humid air being a mixture
            %   of ideal gases.
            %
            %INPUTS:
            %   (double) pressure - current pressure of mixture [Pa]
            %   (double) temp - current temperature of mixture [K]


            %The method assumes the gases, includeing water vapour, are an
            %ideal gas mixture to determine the absolute humidity of the
            %mixture.

            %Determine satruation vapour pressure of water at temp

            if obj.psychro

                %evaluate the saturation water vapour pressure
                satWaterVapPressure =
hacsimulator.PropertyCalculator.evalSatVapPressure(...
                    temp, {'water'}, obj.refpropFlag);

                %evaluate water vapour pressure
                waterVapourPressure = obj.RELATIVE_HUMIDITY * ...
                    satWaterVapPressure;
```

```
                %Determine and set molarAbsHumidity
                obj.molarAbsHumidity = waterVapourPressure /...
                    (pressure - waterVapourPressure);

            else
                obj.molarAbsHumidity = 0;
            end

        end %setMolarAbsHumidity

        function setHumidMolFracs(obj)
            %SETHUMIDMOLFRACS Sets the humidMolFracs in the components
property
            % setHumidMolFracs sets the humidMolFrac properties of the
            % components based on the current value of molarAbsHumidity
            %
            %CALL: obj.setHumidMolFracs()

            for iComponent = 1:1:obj.numComponents

                %get dry mole fraction
                dryMolFrac = obj.components(iComponent).dryMolFrac;

                %evaluate humid mole fraction
                humidMolFrac = dryMolFrac / (1 + obj.molarAbsHumidity);

                %assign value to component
                obj.components(iComponent).setHumidMolFrac(humidMolFrac);

            end

        end %setHumidMolFracs

        function setDryMolMass(obj)
            %SETDRYMOLMASS Sets dryMolMass based on current values of dry mol
fractions
            %
            %CALL: obj.setDryMolMass()

            mixMolMass = 0;

            for iComponent = 1:1:obj.numComponents
                %Get values of mole fraction and molar mass from
obj.components
                molFrac = obj.components(iComponent).dryMolFrac;
                molMass = obj.components(iComponent).molMass;

                %modify mixMolMass
                mixMolMass = mixMolMass + molFrac * molMass;

            end %mixMolMass calc loop
```

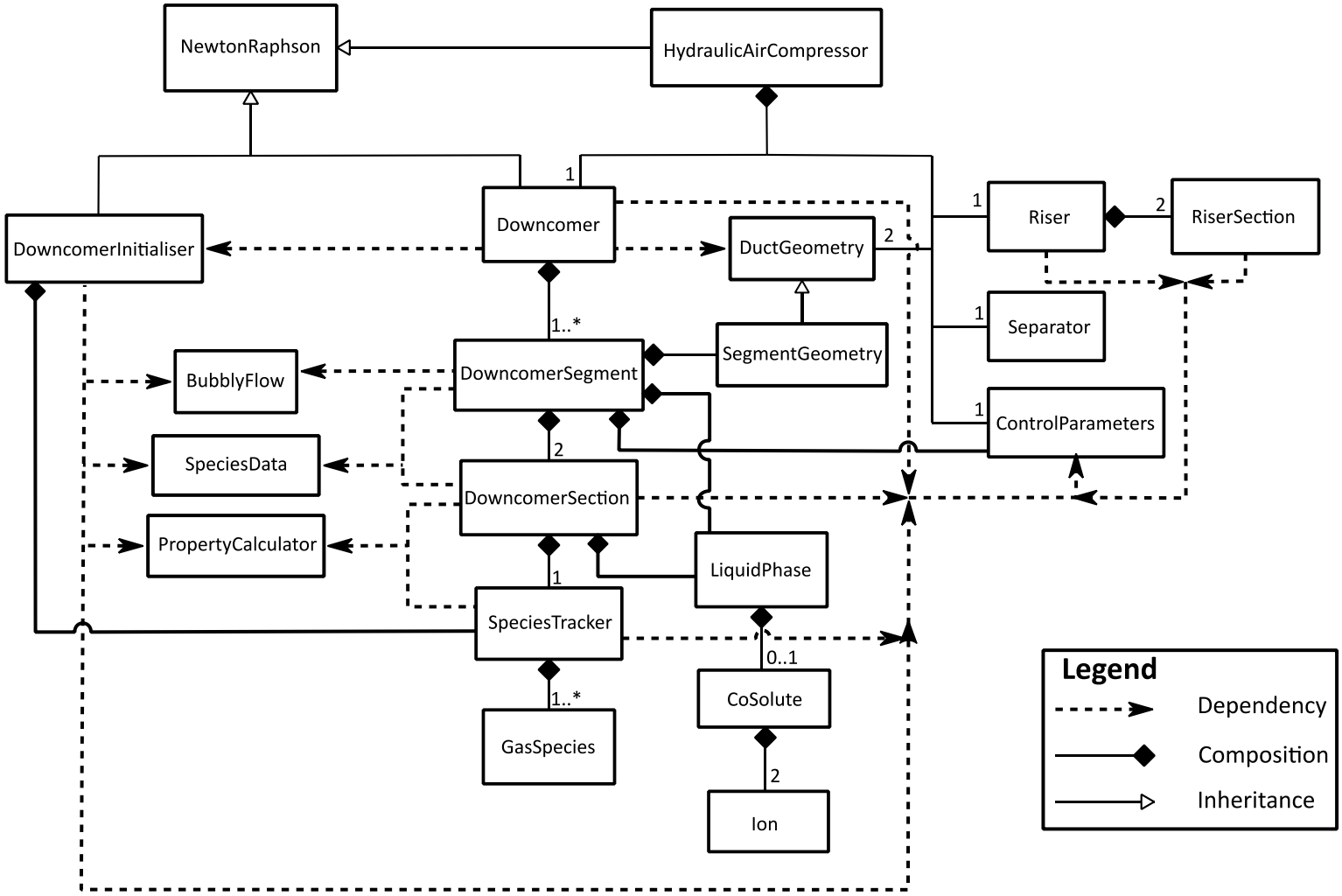
```
            obj.dryMolMass = mixMolMass;

        end %setDryMolMass

        function setHumidMolMass(obj)
            %SETHUMIDMOLMASS Sets humidMolMass based on current values in
humidMassFracs
            %
            %CALL: obj.setHumidMolMass()
            if obj.psychro
                MOL_MASS_H2O = 18e-3;

                humidMolFracArray = obj.getMolFracArray(true);

                molMassArray = obj.getMolMassArray();

                molMassArray = [molMassArray, MOL_MASS_H2O];

                obj.humidMolMass = sum(humidMolFracArray .* molMassArray);
            else

                obj.humidMolMass = obj.dryMolMass;

            end

        end %setHumidMolMass

        function setAbsHumidity(obj)
            %SETABSHUMIDTY Setter method for the absHumidity property
            %   setAbsHumidity Determines and sets the value of
            %   absHumidity based upon the current values of
            %   molarAbsHumidity and dryMolMass.

            MOL_MASS_H2O = 18e-3;

            obj.absHumidity = (MOL_MASS_H2O / obj.dryMolMass) * ...
                obj.molarAbsHumidity;

        end %setAbsHumidity

    end %private methods

end %SpeciesTracker
```

Appendix B: Sample MATLAB profile summary table

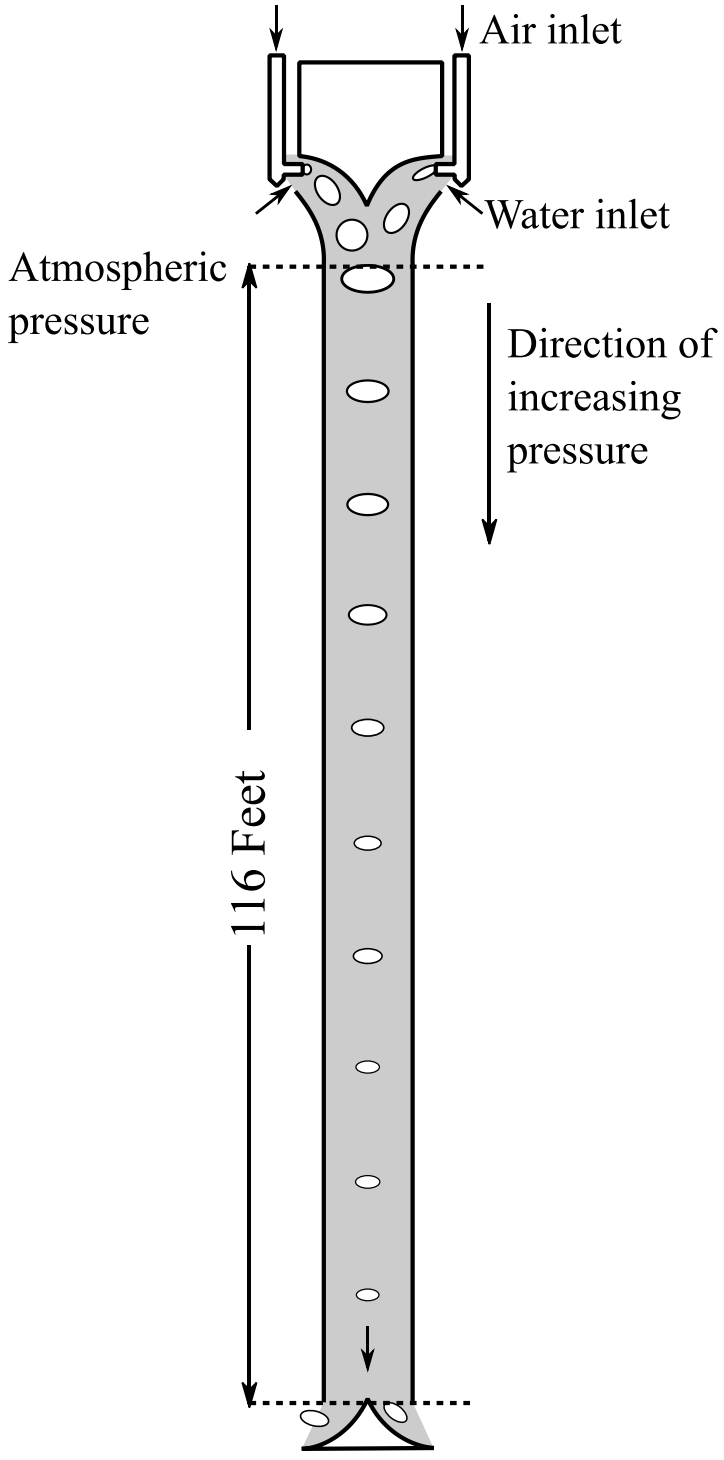
Sample from a profile summary table showing the functions which take the most time in the downcomer model. The function which takes the most time is in bold.

Function Name	Calls	Total Time [s]	Self-Time [s]
RaggedChutesDowncomerGasMixtureTest	1	33.573	0.009
Downcomer>Downcomer.solveSegments	1	33.564	0.038
NewtonRaphson>NewtonRaphson.solveSystem	40	33.238	0.082
...ent>DowncomerSegment.evalResiduals	5,639	33.170	0.214
Downcomer>Downcomer.evalResiduals	5,619	33.156	0.115
PropsSI	39,476	22.303	0.964
CoolPropMATLAB_wrap (MEX-file)	**39,476**	**21.338**	**21.338**
...tion>DowncomerSection.evalGasProps	15,508	18.448	0.330
...gt;PropertyCalculator.evalMixtureProp	15,508	17.579	0.504
...t>DowncomerSegment.evalEnergyResid	5,639	14.954	0.252
...DowncomerSection.evalSpecificEnergies	22,556	14.701	0.347
...t;DowncomerSection.evalInternalEnergy	22,556	14.354	0.148
...DowncomerSection.setSectionProperties	4,230	11.599	0.094
...>DowncomerSection.updateFluidProps	4,211	11.579	0.029
...n>DowncomerSection.evalLiquidProps	15,508	7.263	0.193
...;PropertyCalculator.evalPureFluidProp	15,508	7.070	0.397

Appendix C: HAC simulator simplified class diagram



Appendix D: Dominion Cotton Mills HAC downcomer schematic modified from Taylor (1897)



Appendix E: Gas Yield Data

Atmospheric air

Downcomer length [m]	Gas Species	Gas yield, $\dot{n}_{j,out} / \dot{n}_{j,in}$ [%]				
		500	1,000	2,000	7,000	20,000
1	N2	100.00	100.00	100.00	99.99	99.99
1	O2	100.00	100.00	99.99	99.98	99.97
1	Ar	100.00	100.00	99.99	99.98	99.98
1	CO2	100.02	99.90	99.74	99.45	99.33
10	N2	99.95	99.81	99.61	99.26	99.11
10	O2	99.88	99.56	99.07	98.25	97.90
10	Ar	99.90	99.62	99.21	98.54	98.24
10	CO2	96.83	89.90	81.69	71.67	68.20
20	N2	99.75	99.25	98.55	97.48	97.08
20	O2	99.42	98.25	96.64	94.16	93.22
20	Ar	99.50	98.48	97.10	95.06	94.32
20	CO2	86.86	69.70	56.01	44.33	41.26
50	N2	98.35	95.99	93.06	88.91	87.43
50	O2	96.33	91.25	85.04	76.31	73.18
50	Ar	96.59	91.92	86.49	79.33	76.91
50	CO2	51.18	32.95	24.02	17.78	16.22
100	N2	94.55	88.30	80.67	70.13	66.54
100	O2	88.74	76.83	63.25	45.68	39.84
100	Ar	88.84	77.50	65.25	50.32	45.63
100	CO2	26.37	15.68	10.64	7.07	6.19
200	N2	85.35	70.66	53.31	32.35	26.38
200	O2	72.77	51.02	30.18	10.78	6.35
200	Ar	72.20	51.01	31.27	12.92	8.66
200	CO2	11.62	6.10	3.40	1.55	1.16

Flue gas

Downcomer length [m]	Gas Species	Gas yield, $\dot{n}_{j,out} / \dot{n}_{j,in}$ [%]				
		500	1,000	2,000	7,000	20,000
1	N2	100.00	100.00	100.00	99.99	99.99
1	O2	100.00	100.00	99.99	99.98	99.97
1	Ar	100.00	100.00	99.99	99.98	99.98
1	CO2	100.01	99.88	99.71	99.43	99.31
10	N2	99.93	99.78	99.55	99.18	99.04
10	O2	99.84	99.48	98.94	98.06	97.72
10	Ar	99.87	99.56	99.11	98.38	98.11
10	CO2	96.07	88.42	79.60	69.21	65.98
20	N2	99.69	99.09	98.29	97.14	96.76
20	O2	99.27	97.89	96.03	93.39	92.48
20	Ar	99.37	98.17	96.59	94.42	93.71
20	CO2	84.06	65.28	51.09	39.78	36.99
50	N2	97.82	94.92	91.62	87.39	86.02
50	O2	95.18	88.99	82.07	73.27	70.39
50	Ar	95.53	89.88	83.88	76.70	74.49
50	CO2	43.98	27.33	19.83	14.85	13.64
100	N2	92.73	85.25	77.00	66.80	63.62
100	O2	85.19	71.39	57.25	40.90	35.88
100	Ar	85.36	72.35	59.71	45.84	41.83
100	CO2	20.88	12.32	8.44	5.79	5.16
200	N2	80.74	63.94	46.45	27.95	23.13
200	O2	65.52	42.64	23.62	8.18	4.89
200	Ar	64.96	42.81	24.80	10.04	6.85
200	CO2	8.68	4.51	2.52	1.21	0.93

Appendix F: Co-solute Test Data

Co-solute test results at an inlet temperature of 294.15 K

Mass fraction of co-solute	Outlet mole fraction oxygen, χ' [mol / mol]		
	NaCl	$MgCl_2$	K_2CO_3
0.02	0.1793	0.1797	0.1790
0.04	0.1825	0.1829	0.1819
0.06	0.1855	0.1859	0.1848
0.08	0.1882	0.1888	0.1874
0.1	0.1908	0.1915	0.1899
0.12	0.1931	0.1940	0.1922
0.14	0.1952	0.1962	0.1943
0.16	0.1971	0.1981	0.1962

Mass fraction of co-solute	Outlet concentration of oxygen in liquid, $C_{O2,B}$ [mol/m^3]		
	NaCl	$MgCl_2$	K_2CO_3
0.02	1.4213	1.4084	1.4396
0.04	1.2908	1.2784	1.3244
0.06	1.1663	1.1541	1.2111
0.08	1.0486	1.0325	1.1015
0.1	0.9382	0.9147	0.9963
0.12	0.8350	0.8040	0.8961
0.14	0.7391	0.7033	0.8013
0.16	0.6510	0.6127	0.7122

Co-solute test results at an inlet temperature of 308.15 K

Mass fraction of co-solute	Outlet mole fraction oxygen, χ' [mol / mol]		
	NaCl	$MgCl_2$	K_2CO_3
0.02	0.1834	0.1836	0.1831
0.04	0.1861	0.1863	0.1857
0.06	0.1887	0.1889	0.1881
0.08	0.1910	0.1913	0.1904
0.1	0.1932	0.1935	0.1926
0.12	0.1952	0.1956	0.1945
0.14	0.1970	0.1975	0.1964
0.16	0.1986	0.1991	0.1980

Mass fraction of co-solute	Outlet concentration of oxygen in liquid, $C_{O2,B}$ [mol/m^3]		
	NaCl	$MgCl_2$	K_2CO_3
0.02	1.2693	1.2604	1.2823
0.04	1.1594	1.1528	1.1826
0.06	1.0545	1.0507	1.0848
0.08	0.9549	0.9505	0.9902
0.1	0.8610	0.8529	0.8991
0.12	0.7726	0.7598	0.8122
0.14	0.6903	0.6732	0.7299
0.16	0.6138	0.5938	0.6523

Appendix G: Gas-liquid separator underflow Sauter mean bubble diameter worked example

The following is a worked example to evaluate the Rosin-Rammler mean bubble diameter at the gas-liquid separator inlet and the Sauter mean bubble diameter at the underflow with the following inputs:

Sauter mean diameter at separator inlet	2.00E-03	m
Rosin-Rammler spread parameter	2.5	
Volume flow rate of air at separator inlet	0.1	m^3/s
Separation Efficiency	99.00	%

The Rosin-Rammler mean is initialized with the Sauter mean and after iteratively solving for the Rosin-Rammler mean which gives the following results:

Rosin-Rammler mean	1.47E-03m
Sauter Mean Diameter check	2.00E-03m
Residual	8.73E-11

with the following bubble size distribution

Edge No.	Edges	Diameter [m]	Cumulative fraction	Fraction	$\Delta\mu_2'$ [m^2]	$\Delta\mu_3'$ [m^3]
-7	8.476E-03					
-6	6.601E-03	7.499E-03	100.00%	2.52E-14	1.417E-18	1.063E-20
-5	5.141E-03	5.840E-03	100.00%	5.26E-08	1.795E-12	1.048E-14
-4	4.004E-03	4.548E-03	100.00%	1.27E-04	2.627E-09	1.195E-11
-3	3.118E-03	3.542E-03	99.99%	8.09E-03	1.015E-07	3.595E-10
-2	2.428E-03	2.759E-03	99.18%	6.83E-02	5.198E-07	1.434E-09
-1	1.891E-03	2.149E-03	92.35%	1.76E-01	8.131E-07	1.747E-09
0	1.473E-03	1.673E-03	74.74%	2.26E-01	6.333E-07	1.060E-09
1	1.147E-03	1.303E-03	52.12%	1.95E-01	3.318E-07	4.324E-10
2	8.933E-04	1.015E-03	32.58%	1.36E-01	1.396E-07	1.417E-10
3	6.957E-04	7.904E-04	19.02%	8.34E-02	5.212E-08	4.119E-11
4	5.418E-04	6.156E-04	10.68%	4.81E-02	1.824E-08	1.123E-11
5	4.220E-04	4.794E-04	5.87%	2.68E-02	6.164E-09	2.955E-12
6	3.286E-04	3.734E-04	3.18%	1.47E-02	2.045E-09	7.634E-13
7	2.559E-04	2.908E-04	1.72%	7.94E-03	6.715E-10	1.953E-13
8	1.993E-04	2.265E-04	0.92%	4.28E-03	2.194E-10	4.967E-14
9	1.552E-04	1.764E-04	0.49%	2.30E-03	7.145E-11	1.260E-14
10	1.209E-04	1.374E-04	0.27%	1.23E-03	2.324E-11	3.192E-15
11	9.415E-05	1.070E-04	0.14%	6.60E-04	7.551E-12	8.078E-16
12	7.333E-05	8.331E-05	0.08%	3.53E-04	2.453E-12	2.043E-16
13	5.711E-05	6.488E-05	0.04%	1.89E-04	7.965E-13	5.168E-17
14	4.448E-05	5.053E-05	0.02%	2.18E-04	5.565E-13	2.812E-17
Σ				1.00	**2.621E-06**	**5.243E-09**

Now the cut size of the separator can be determined, the underflow bubble size distribution simulated and the Sauter mean bubble diameter evaluated assuming the same spread parameter as the inlet.

Inlet Rosin-Rammler mean diameter	1.47E-03	m
Rosin-Rammler spread parameter	2.5	
Separation efficiency	99.00%	
Separator cut-size	**2.34E-04**	**m**
Separator underflow	**0.001**	**m^3/s**

Using the cut-size as the Rosin-Rammler mean of the underflow and the same spread parameter as the inlet produces the following size distribution:

Edge No.	Edges	Diameter [m]	Cumulative fraction	Fraction	$\Delta\mu_2'$ [m^2]	$\Delta\mu_3'$ [m^3]
-7	1.346E-03					
-6	1.048E-03	1.191E-03	100.00%	2.52E-14	3.574E-20	4.257E-23
-5	8.164E-04	9.275E-04	100.00%	5.26E-08	4.526E-14	4.198E-17
-4	6.358E-04	7.223E-04	100.00%	1.27E-04	6.625E-11	4.786E-14
-3	4.952E-04	5.626E-04	99.99%	8.09E-03	2.559E-09	1.440E-12
-2	3.856E-04	4.381E-04	99.18%	6.83E-02	1.311E-08	5.744E-12
-1	3.003E-04	3.412E-04	92.35%	1.76E-01	2.051E-08	6.996E-12
0	2.339E-04	2.657E-04	74.74%	2.26E-01	1.597E-08	4.244E-12
1	1.822E-04	2.070E-04	52.12%	1.95E-01	8.369E-09	1.732E-12
2	1.419E-04	1.612E-04	32.58%	1.36E-01	3.521E-09	5.675E-13
3	1.105E-04	1.255E-04	19.02%	8.34E-02	1.314E-09	1.650E-13
4	8.605E-05	9.776E-05	10.68%	4.81E-02	4.599E-10	4.496E-14
5	6.701E-05	7.613E-05	5.87%	2.68E-02	1.554E-10	1.183E-14
6	5.219E-05	5.929E-05	3.18%	1.47E-02	5.157E-11	3.057E-15
7	4.065E-05	4.618E-05	1.72%	7.94E-03	1.694E-11	7.820E-16
8	3.165E-05	3.596E-05	0.92%	4.28E-03	5.532E-12	1.989E-16
9	2.465E-05	2.801E-05	0.49%	2.30E-03	1.802E-12	5.047E-17
10	1.920E-05	2.181E-05	0.27%	1.23E-03	5.860E-13	1.278E-17
11	1.495E-05	1.699E-05	0.14%	6.60E-04	1.904E-13	3.235E-18
12	1.165E-05	1.323E-05	0.08%	3.53E-04	6.186E-14	8.184E-19
13	9.069E-06	1.030E-05	0.04%	1.89E-04	2.009E-14	2.070E-19
14	7.063E-06	8.024E-06	0.02%	2.18E-04	1.404E-14	1.126E-19
Σ				**1.00**	**6.611E-08**	**2.100E-11**

Then the Sauter mean bubble diameter and bubble flux of the underflow is calculated.

Sauter Mean Diameter	**3.176E-04**	**m**
Bubble volume	1.678E-11	m^3
Bubble flux	**5.9607E+07**	**1/s**